91602
new 82-10

COMPLIMENTARY COPY
Please Credit Ontario Hydro Research if
Reproduced in Whole or in Part

LEGAL NOTICE

This report was prepared by Ontario Hydro and, while effort has been made to assure the accuracy and applicability of the material contained herein, neither Ontario Hydro nor any person acting on its behalf:

(a) makes any warranty, express or implied with respect to the use of any information, apparatus, method or process disclosed in this report or that such use may not infringe privately owned rights; or

(b) assumes any liabilities with respect to the use of, or for damages resulting from the use of, any information, apparatus, method or process disclosed in this report.

ontario hydro
research division

3D PARTICLE TRACKING ALGORITHM TO MODEL RADIONUCLIDE TRANSPORT

Report No. 91-161-K

Arthur H.C. Chan & Amrit L. Punhani
Mathematicians
Statistical Methods & Analysis Unit
Operations Research Department

Abstract

To assess the suitability of a rock mass as a waste repository host, it is important to determine how long it will take for radionuclides to travel through the rock mass. An algorithm for a three dimensional tracking program is presented and compared to that developed by the Whiteshell Laboratories. The results obtained using a program developed from this algorithm are then discussed. A user's manual is also provided.

Key Words: Radionuclide transport, geosphere, finite element method, particle tracking.

Work done for
Whiteshell Laboratories
Atomic Engery of Canada Limited
Under the Technical Asistance Program

job	file	date	report no.
7406-72-167	809.7	June 28, 1991	91-161-K

TABLE OF CONTENTS

Section		Page
1. INTRODUCTION		1
2. THE OSCILLATION PROBLEMS		1
3. THE SOLUTION		2
4. RESULTS AND DISCUSSION		2
ACKNOWLEDGEMENTS		3
REFERENCES		4
TABLE 1	Particle Tracking Results - 100m Well	5
TABLE 2	Particle Tracking Results - 200m Well	7
APPENDIX I	Element Step Algorithm	8
APPENDIX II	FORTRAN Source Code	14
APPENDIX III	System Manual	57
APPENDIX IV	User's Manual	90

ontario hydro
research division

To Dr. D. Mills
Director of Research
Research Division
KR 116

3D PARTICLE TRACKING ALGORITHM TO MODEL RADIONUCLIDE TRANSPORT

1. INTRODUCTION

In the safety assessment of a nuclear waste repository site, it is necessary to determine the time it takes for the radio-nuclides to travel from the repository to outside the host rock mass. A radionuclide particle travelling through the rock mass via the hydraulic gradient of the ground-water is traced. The finite element mathematical method is used to trace this path.

Dr. Tin Chan of the Whiteshell Laboratories (WL) of the Atomic Energy of Canada Ltd. (AECL) has been investigating this problem [1]. He detected some aberrations that are not indigenous to the movement of the particles, but originated from the numerical algorithm.

In this report, the problems arising from the existing finite element algorithm used by WL are outlined. Then, the rationale behind the modifications to resolve these problems will be given. Finally, the results of the modifications, and comparison of the results with those from the existing code will be given. Appendices include the FORTRAN source code and the user's manual for these programs.

2. THE OSCILLATION PROBLEMS

In the WL algorithm, the particle was followed through the mesh in steps of a fixed time duration. The velocity gradient for each move was taken to be that existing in the element where the move was started. Call this method the *time step* algorithm.

The elements are artificially created partitions and, as such, can sometimes have two adjacent elements with velocity gradients that oppose each other. In such a case, the simulation may cause the particle to jump back and forth between these two elements repeatedly. The path traced in such a situation is a zig-zag that can take considerably longer to traverse. This leads to oscillations in the time step algorithm.

3. THE SOLUTION

In order to resolve the oscillation problem, and the desire to provide independent confirmation of the WL results, a different algorithm is proposed. This proposed algorithm traces the particle in motion from one boundary of an element to another boundary of the same element according to the velocity gradient existing in that element. The next move is then into the appropriate adjacent element. This is called the *element step* algorithm.

At each boundary the validity of moving into the next element is considered. If the velocity gradient in the next element is opposed to that in the last element, the particle would necessarily be pinned up against the boundary as motion across the next element would be impossible. In such a case it is necessary to travel along the boundary between the elements. This then eliminates oscillations between the elements.

The mathematical background and detailed description of this algorithm is given in Appendix I. In Appendix II, the computer source code is given. In appendices, III and IV, the system manual and the user's manual for this algorithm are given, respectively.

4. RESULTS AND CONCLUSIONS

The element step algorithm eliminates the oscillations encountered in the time step algorithm of WL. As a result, the time required to travel through the rock mass is shorter than that obtained by WL when oscillations occured. This can be intepreted that the new algorithm produces a more conservative estiamte of the travelling time. In the case where there is no oscillation, the two algorithms produce similar results. Tables 1 and 2 provide a comparison between the results of two algorithms. Finally, the WL algorithm usually takes 600 steps to complete each simulation whereas the element step algorithm takes only 20 steps. Consequently, the computational time is much shorter.

ACKNOWLEDGEMENTS

The authors would like to thank Dr. Tin Chan of WL for providing information and suggestions regarding the time step algorithm. The authors would also like to

acknownledge the contribution of Mr. Chris Black in the software development of the element step algorithm.

Approved by:

A. H. Chung
Manager
Operations Research Department

Prepared by:

A.H. Chan
Mathematician
Statistical Methods & Analysis Unit
Operations Research Department

A.L. Punhani
Senior Mathematician
Statistical Methods & Analysis Unit
Operations Research Department

REFERENCES

1. Nakka, B.W., and Chan, T., A Particle Tracking Code (TRACK3D) for Geosphere Flow Modelling: Descriptions and User's Manual , to appear in 1991 as an AECL report.

2. Korn, G.A., and Korn, T.M., Mathematical Handbook for Scientists and Engineers, second edition, McGraw-Hall Book Company, New York, 1968.

TABLE 1

PARTICLE TRACKING RESULTS
100 METRE WELL

Element number	Initial location x (m)	y (m)	z (m)	Exit location x (m)	y (m)	z (m)	Travel Time (yrs) Hydro	Travel Time (yrs) AECL
8813	195.83	-1000.00	-25.88	-236.84	-739.16	255.14	462.9	476.0
8844	195.83	-975.00	-25.88	-236.84	-714.16	255.07	462.8	473.0
8844	195.83	-950.00	-25.88	-236.84	-689.16	255.00	462.8	472.0
8844	195.83	-925.00	-25.88	-236.84	-664.16	254.93	462.7	472.0
8844	195.83	-900.00	-25.88	-314.44	-350.00	153.67	501.8	502.0
8844	195.83	-875.00	-25.88	-314.44	-350.00	153.67	485.6	462.0
8844	195.83	-850.00	-25.88	-316.15	-350.00	142.05	426.2	440.0
8844	195.83	-825.00	-25.88	-314.11	-385.87	139.92	401.6	410.0
8844	195.83	-800.00	-25.88	-314.23	-388.18	139.71	382.8	392.0
8844	195.83	-775.00	-25.88	-314.35	-390.48	139.50	364.0	375.0
8844	195.83	-750.00	-25.88	-314.37	-392.64	139.48	344.5	355.0
8844	195.83	-725.00	-25.88	-314.05	-394.38	140.02	322.7	366.0
8844	195.83	-700.00	-25.88	-313.73	-396.12	140.55	301.0	314.0
8844	195.83	-675.00	-25.88	-313.72	-397.24	140.57	284.7	296.0
8844	195.83	-650.00	-25.88	-313.74	-397.56	140.54	265.9	277.0
8910	195.83	-625.00	-25.88	-313.71	-400.46	140.60	255.7	264.0
8910	195.83	-600.00	-25.88	-313.67	-403.36	140.67	245.5	255.0
8910	195.83	-575.00	-25.88	-309.90	-353.81	138.92	239.2	242.0
8910	195.83	-550.00	-25.88	4.03	-546.42	49.28	149.9	236.0
8910	195.83	-525.00	-25.88	-321.56	-350.00	141.70	229.5	232.0
8910	195.83	-500.00	-25.88	-321.34	-350.00	142.07	227.5	230.0
8910	195.83	-475.00	-25.88	-321.21	-350.00	142.30	225.2	228.0
8910	195.83	-450.00	-25.88	-321.13	-350.00	142.43	222.1	225.0
8910	195.83	-425.00	-25.88	-321.13	-350.00	142.43	218.1	221.0
8910	195.83	-400.00	-25.88	-321.03	-350.00	142.60	214.9	217.0
8910	195.83	-375.00	-25.88	-320.92	-350.00	142.78	211.7	217.0
8910	195.83	-350.00	-25.88	-320.61	-350.00	143.29	205.2	214.0
8921	195.83	-325.00	-25.88	-320.55	-350.00	143.40	205.2	212.0
8921	195.83	-300.00	-25.88	-79.11	-350.26	67.91	147.2	215.0
8921	195.83	-275.00	-25.88	-320.22	-350.00	143.96	208.9	216.0
8921	195.83	-250.00	-25.88	-320.11	-350.00	144.14	208.9	214.0
8921	195.83	-225.00	-25.88	-319.74	-350.00	144.77	210.4	214.0
8921	195.83	-200.00	-25.88	-319.53	-350.00	145.11	211.7	215.0
8921	195.83	-175.00	-25.88	-319.33	-350.00	145.45	213.0	217.0
8921	195.83	-150.00	-25.88	-319.12	-350.00	145.80	214.3	217.0
8921	195.83	-125.00	-25.88	-318.92	-350.00	146.15	215.5	218.0
8921	195.83	-100.00	-25.88	-318.71	-350.00	146.49	216.8	219.0
8921	195.83	-75.00	-25.88	-318.48	-350.00	146.87	218.1	220.0
8921	195.83	-50.00	-25.88	-318.12	-350.00	147.49	219.4	221.0
8936	195.83	-25.00	-25.88	-317.65	-350.00	148.27	223.5	226.0
8936	195.83	0.00	-25.88	-320.70	-350.00	148.97	228.3	231.0
8936	195.83	25.00	-25.88	-320.82	-349.92	148.63	234.1	237.0

TABLE 1 (Cont'd)

Element number	Initial location x (m)	y (m)	z (m)	Exit location x (m)	y (m)	z (m)	Travel Time (yrs) Hydro	Travel Time (yrs) AECL
8936	195.83	50.00	-25.88	-311.52	-299.98	144.30	238.9	247.0
8943	195.83	75.00	-25.88	-312.04	-305.43	143.42	250.0	257.0
8943	195.83	100.00	-25.88	-312.62	-311.22	142.45	261.5	269.0
8943	195.83	125.00	-25.88	-313.21	-317.02	141.44	272.9	281.0
8943	195.83	150.00	-25.88	-320.75	-336.95	140.03	284.2	290.0
8943	195.83	175.00	-25.88	-320.19	-334.11	140.98	295.3	301.0
8943	195.83	200.00	-25.88	-320.71	-339.90	140.11	306.4	313.0
8943	195.83	225.00	-25.88	-319.88	-331.67	139.56	317.3	323.0
8943	195.83	250.00	-25.88	-309.32	-293.52	136.06	325.4	335.0
8943	195.83	275.00	-25.88	-321.42	-331.55	138.09	343.7	344.0
8943	195.83	300.00	-25.88	-300.18	-257.07	125.31	348.0	366.0

TABLE 2

PARTICLE TRACKING RESULTS
200 METRE WELL

Element number	Initial location x (m)	y (m)	z (m)	Exit location x (m)	y (m)	z (m)	Travel Time (yrs) Hydro	Travel Time (yrs) AECL
4386	837.68	-1000.00	-239.40	-213.52	-922.40	255.54	1252.5	1271.0
6419	837.68	-950.00	-239.40	-216.43	-915.61	255.54	1192.1	1209.0
6419	837.68	-900.00	-239.40	-217.70	-896.98	255.49	1133.7	1140.0
6419	837.68	-850.00	-239.40	-217.71	-846.98	255.35	1133.7	1140.0
6419	837.68	-800.00	-239.40	-217.71	-796.99	255.21	1133.6	1139.0
6419	837.68	-750.00	-239.40	-217.72	-746.99	255.06	1133.6	1140.0
6419	837.68	-700.00	-239.40	-217.72	-697.00	254.92	1133.5	1140.0
6419	837.68	-650.00	-239.40	-222.45	-650.05	254.81	1132.3	1140.0
6401	837.68	-600.00	-239.40	-247.30	-650.02	254.94	1117.5	1154.0
6401	837.68	-550.00	-239.40	-252.16	-650.01	254.97	1084.8	1130.0
6401	837.68	-500.00	-239.40	-252.51	-650.01	254.97	1045.3	1107.0
6401	837.68	-450.00	-239.40	-253.17	-650.01	254.97	1037.7	1079.0
6401	837.68	-400.00	-239.40	-302.08	-626.79	255.04	1085.7	1095.0
6401	837.68	-350.00	-239.40	-302.50	-577.54	255.12	1086.1	1095.0
6383	837.68	-300.00	-239.40	-302.67	-528.87	255.20	1086.3	1095.0
6383	837.68	-250.00	-239.40	-308.54	-504.81	255.24	1093.2	1098.0
6383	837.68	-200.00	-239.40	-307.19	-472.89	255.30	1091.1	1097.0
6383	837.68	-150.00	-239.40	-306.53	-444.31	255.34	1088.6	1095.0
6383	837.68	-100.00	-239.40	-306.58	-418.70	255.39	1085.3	1091.0
6383	837.68	-50.00	-239.40	-306.40	-396.95	255.42	1079.7	1085.0
6365	837.68	0.00	-239.40	-292.86	-372.18	255.46	42873.7	47032.0
6374	837.68	50.00	-239.40	-362.02	-354.39	255.49	1123.8	1127.0
6374	837.68	100.00	-239.40	-430.85	-332.34	255.25	1175.9	1181.0
6374	837.68	150.00	-239.40	-395.51	-293.25	255.40	1155.2	1162.0
6374	837.68	200.00	-239.40	-395.04	-262.96	255.40	1161.6	1167.0
6374	837.68	250.00	-239.40	-398.23	-234.86	255.39	1170.6	1179.0
6374	837.68	300.00	-239.40	-400.56	-207.82	255.38	1178.3	1185.0
6392	837.68	350.00	-239.40	-405.20	-188.42	255.36	1199.3	1208.0
6392	837.68	400.00	-239.40	-401.67	-167.49	255.38	1210.8	1217.0
6392	837.68	450.00	-239.40	-392.62	-146.83	255.41	1217.3	1224.0
6392	837.68	500.00	-239.40	-381.89	-127.10	255.46	1221.4	1230.0
6392	837.68	550.00	-239.40	-380.29	-112.80	255.47	1236.2	1250.0
6392	837.68	600.00	-239.40	-380.86	-99.37	255.47	1252.0	1265.0
6410	837.68	650.00	-239.40	-377.34	-87.12	255.48	1277.5	1292.0
6410	837.68	700.00	-239.40	-371.07	-74.73	255.51	1298.3	1311.0
6410	837.68	750.00	-239.40	-362.00	-61.96	255.55	1313.0	1324.0
6410	837.68	800.00	-239.40	-360.60	-50.00	255.55	1323.0	1342.0
6410	837.68	850.00	-239.40	-367.29	-36.27	255.52	1353.0	1364.0
6410	837.68	900.00	-239.40	-370.10	-23.73	255.51	1384.8	1401.0
6410	837.68	950.00	-239.40	-372.62	-12.25	255.50	1415.5	1426.0
1007	837.68	1000.00	-239.40	-375.54	-4.69	255.49	1457.1	1469.0

APPENDIX I

Element Step Algorithm

1. Basic Formulae in Geometry

The following are some basic terminologies in analytical geometry and vector analysis needed in the subsequent discussions.

(a) **Inner product of two vectors**

Let $\mathbf{v} = (v_1,v_2,v_3)$ and $\mathbf{u} = (u_1,u_2,u_3)$ be two vectors in a 3D space. Then inner product $< , >$ is defined by

$$<\mathbf{v},\mathbf{u}> = v_1u_1 + v_2u_2 + v_3u_3$$

(b) **The cross product of two vectors**

Let $\mathbf{v}$ and $\mathbf{u}$ be defined as in (a). Then the cross product of $\mathbf{v}$ and $\mathbf{u}$ is

$$\mathbf{v}\times\mathbf{u} = \begin{bmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ v_1 & v_2 & v_3 \\ u_1 & u_2 & u_3 \end{bmatrix}$$

where $\mathbf{i}$, $\mathbf{j}$ and $\mathbf{k}$ are the unit vectors of the coordinates.

(c) **Norm of a vector**

Then norm of a vector $\mathbf{v} = (v_1,v_2,v_3)$ is defined by

$$\|\mathbf{v}\| = \sqrt{v_1^2 + v_2^2 + v_3^2}$$

(d) **Normal vector of a plane**

A plane in a 3D space can be defined by three point in the space. Suppose that we have three points $x_i = (x_i,y_i,z_i)$, i=1,2,3 in the space. Define two vectors

$$\mathbf{v} = (x_1-x_2, y_1-y_2, z_1-z_2) \quad \text{and} \quad \mathbf{u} = (x_3-x_2, y_3-y_2, z_3-z_2)$$

Then, the normal vectors is defined by

$$\mathbf{n} = \mathbf{v}\times\mathbf{u}$$

From the definition of cross product, the direction of **n** is reverse by using **u**×**v**.

2. Formulae for determining particle location and travel time

The formulae in this section can be found in many vector analysis and analytical geometry text books, for example [2].

2.1 Time required to travel from a point to a plane

Let $\mathbf{x}_0 = (x_0, y_0, z_0)$ be a point in space and $ax + by + cz = d$ be the equation of a hyper-plane. Let **n** denote the normal vector of the plane which is pointing away from the point $\mathbf{x}_0$. Let $\mathbf{v} = (v_1, v_2, v_3)$ denote the velocity vector of a particle **P**. The time t that **P** needed to travel from $\mathbf{x}_0$ to the plane is

$$t = |d - ax_0 + by_0 + cz_0| / \langle \mathbf{v}, \mathbf{n} \rangle$$

If $t \leq 0$, it means the velocity vector is point away from the plane, and **P** will not reach the plane.

2.2 Time required to travel from a point to a line on the same plane with a given velocity vector

The formula of a line **L** in space is:

$$\frac{(x-a)}{A} = \frac{(y-b)}{B} = \frac{(z-c)}{C}$$

The distance from a point $\mathbf{x}_0 = (x_0, y_0, z_0)$ to the line **L** is

$$d(\mathbf{P},\mathbf{L}) = \left[\frac{\begin{vmatrix} y_0 - b & z_0 - c \\ B & C \end{vmatrix}^2 + \begin{vmatrix} z_0 - c & x_0 - a \\ C & A \end{vmatrix}^2 + \begin{vmatrix} x_0 - a & y_0 - b \\ A & B \end{vmatrix}^2}{A^2 + B^2 + C^2} \right]^{\frac{1}{2}}$$

Since the **v**, **x** and **L** are on the same plane, the following equation must be satisfied:

$$\begin{vmatrix} x_o - a & y_o - b & z_o - c \\ v_1 & v_2 & v_3 \\ A & B & C \end{vmatrix} = 0$$

The distance l travel from x to L in the direction of v is

$$l = d(P,L)\sin(\cos^{-1}[Av_1 + Bv_2 + Cv_3])$$

The time t needed to travel the distance l is

$$t = l/||v||$$

2.3 The location, x', of a particle after travel time t from x, is

$$x' = (x_1 + tv_1, x_2 + tv_2, x_3 + tv_3)$$

3. Notations

1. Let **P** denote the particle to be traced and $x = (x_1,x_2,x_3)$ denote its location.
2. Let **e** denote an element in the mesh
3. Let $v(e) = (v(e)_1,v(e)_2,v(e)_3)$ denote the associated velocity vector of **e**.
4. Each element has six (6) sides and let $S_{e1}, \ldots, S_{e6}$ denote the 6 sides of **e**.
5. Each side has one unique adjacent element. Let $\varphi(S_{ei})$ denote of adjacent element of S_{ei}.
6. Each side has four (4) edges and four nodes. Let $\varepsilon(e)_{ij}$ and $\eta(e)_{ij}$ be the jth edge and jth node of side S_{ei} of element **e**, respectively.
7. Each edge can have up to 60 elements attached to it. Let $\psi(\varepsilon(e)_{ij},k)$ denote the kth element adjacent to edge $\varepsilon(e)_{ij}$.
8. Let t_i denote the time required to travel from x to S_{ei}.

4. The Algorithm

4.1. Particle is inside, or on a side of an element but not on an edge or a node

Let **e** be the element in which the particle **P** is situated. Consider only the situation when **P** is on a side S_{ei}. The case when the particle is inside the element, the solution is trivial. And, let S_{ei} be adjacent to $\varphi(S_{ei})$

Case 1 If the inner product $\langle v(\mathbf{e}), v(\varphi(S_{ei}))\rangle$ is positive, the particle travels through the adjacent element. Then, determine the side which requires the shortest time except the side which the particle is already on. Then move the particle to its next location. This can be accomplished by formulae 1 and 3.

Case 2 If the inner product is negative, the particle cannot travel through the adjacent element. Then, the particle is forced to travel on the side until it reaches an edge.

Case 3 If the inner product is zero, then the particle will stop travelling.

4.2 Particle is on an edge but not on a node

Case 4 **P** travelled an element **e** or along one of its sides S_{ei} before arrived at an edge $\varepsilon(\mathbf{e})_{ij}$ with velocity **v**. Then **P** will travel to one of the $\psi(\varepsilon(\mathbf{e})_{ij},k)$'s if $\langle \mathbf{v}, \mathbf{v}(\psi(\varepsilon(\mathbf{e})_{ij},k))\rangle$ is positive.

Case 5 If all $\langle \mathbf{v}, \mathbf{v}(\psi(\varepsilon(\mathbf{e})_{ij},k))\rangle$ are negative, then P travels along $\varepsilon(\mathbf{e})_{ij}$. The edge has two nodes, say $x_1 = (x_1,y_1,z_1)$ and $x_2 = (x_2,y_2,z_2)$. Define the vector

$$\mathbf{u} = (x_1 - x_2)/\,||x_1 - x_2||$$

Then **P** travels $\varepsilon(\mathbf{e})_{ij}$ with linear velocity of

$$\max\{\, |\langle \mathbf{u}, \mathbf{v}(\psi(\varepsilon(\mathbf{e})_{ij},k))\rangle| :\ k = 1,2, \ldots, K\}$$

The particle will arrive at x_1 if $\langle \mathbf{u}, \mathbf{v}(\psi(\varepsilon(\mathbf{e})_{ij},k))\rangle$ is positive; otherwise, it will go to x_2.

Case 6 If Cases 4 and 5 are not possible, then **P** stops.

4.3 Particle is on a node

Case 7 P arrived at the node from Case 1 or Case 4. Then this is reduced to Case 1.

Case 8 P arrived at the node $\eta(\mathbf{e})_{ij}$ from Case 5. Let $\mathbf{e}_m$, m=1,2,...,M, be the elements which connected to $\eta(\mathbf{e})_{ij}$ other than $\psi(\varepsilon(\mathbf{e})_{ij},k)$'s. Then P will travel to element $\mathbf{e}_m$ if $|\langle \mathbf{u},\mathbf{v}(\mathbf{e}_m)\rangle|$ is maximum.

Case 9 If Cases 7 and 8 are not possible, then P stops.

APPENDIX II

FORTRAN Source Code

```
C ---------------------------------------------------------------
C ** HEADER.F **
C    Contains declarations of COMMON storage areas; external
C    functions and procedures; variables for main program.
C ---------------------------------------------------------------

      include "max.f"
      common /ELEMLST/ elemvert(MAXELEMS,8), elemvel(MAXELEMS,3)
      common /NODELST/ nodelist(MAXNODES,0:16)
      common /NETNODE/ netside(6,5), netused(6,4), netedge(12,2),
     1                 netvel(3)
      common /ELNET/ elside(-1:MAXELEMS, 6, 5),
     1               elused(-1:MAXELEMS,6,4),
     2               eledge(-1:MAXELEMS, 12, 2),
     3               elvel(-1:MAXELEMS, 3)
      common /NODES/ nodes(MAXNODES, 3)
      common /LIST/  nolstarr(MAXNODES, 0:16)
      common /FILES/ elemf, velf, elemnetf, nodelstf,
     1               nodef, sumf, outf, inf
      integer elemvert, nodelist, netside, netused, netedge
      double precision elemvel, netvel, elvel
      integer elside, elused, eledge, nolstarr
      double precision nodes
      character*30 configf, elemf, velf, elemnetf, nodelstf,
     1             nodef, sumf, outf, inf

      external rdconfig, setupsum, rdnodeco, rdnolst, runelem,
     1         rdnewvel, readinit, prbanner, alongedg, dot,
     2         magnitud, chkedge, chknode, thruelem, alongsid,
     3         getelem, getside, elemfr, sidefr, getedge

      integer i, elemid, lastelem, edge, presnode, endside, nextelem,
     1        exitside, startside, nxtelem, exitedge, bestelem,
     2        bestnext, bsnxedge, mainelem, endedge, nextel, exitnode,
     3        nodearr1(0:16), nodearr2(0:16), sumid

      double precision lasttime, startpt(3), vperiod, tottime, endpt(3),
     1       travtime, oldvel(3), exitpt(3), vel(3), curlarge, t,
     2       largemag, bestvel(3), dot, magnitud, vector(3), totdist,
     3       sumstart(3)

      logical more, atnode, atedge, escape, blocked, firstel, chkedge,
     1        chknode, thruelem, alongsid, getelem, getside, elemfr,
     2        sidefr, getedge

      integer X,Y,Z,NODE1,NODE2,ID
      double precision POINTERR
      parameter (X=1,Y=2,Z=3,NODE1=1,NODE2=2,NODE3=3,NODE4=4,ID=5)
      parameter (POINTERR=1D-3)
```

```
C Program          : 3D Particle Tracking System, FORTRAN77 version
C Program Segment  : FUNCTION.F
C Description      : This program segment contains all functions and
C                    subroutines used by the program.
C Last Revision    : February 28, 1991
C Translated by    : Vanessa Williams
C                    SOMAPRO Ltd.
C Authors          : Arthur Chan, Christopher Black, Peter Ng
C ------------------------------------------------------------------
C ------------------------------------------------------------------
      logical function ixedge(startpt,vel,node1,node2,t,endpt)

C     This function tests the edge defined by the given nodes to see
C     if the given velocity vector intersects that edge between the
C     nodes. If it does the value .TRUE. is returned, along with the
C     intersection point and the time to intersection. If it does not,
C     the value .FALSE. is returned. Note that the smallest positive
C     time to intersection is assumed to be the best.
C ------------------------------------------------------------------

      external min, max
      parameter (X=1, Y=2, Z=3, POINTERR=1.0E-03)
      double precision startpt(3), vel(3), node1(3), node2(3),
     1                 endpt(3),t
      double precision a, b, c, time1, time2, time3

      ixedge = .FALSE.
      t = 0.0
      time1 = 0.0
      time2 = 0.0
      time3 = 0.0

      a = node2(X)-node1(X)
      b = node2(Y)-node1(Y)
      c = node2(Z)-node1(Z)

      if (.NOT.(a*vel(Z) .EQ. c*vel(X))) then
         time1=(a*(node1(Z)-startpt(Z))+(startpt(X)-node1(X))*c)/
     1          (a*vel(Z)-c*vel(X))
      endif
      if (.NOT.(b*vel(Z) .EQ. c*vel(Y))) then
         time2=(b*(node1(Z)-startpt(Z))+(startpt(Y)-node1(Y))*c)/
     1          (b*vel(Z)-c*vel(Y))
      endif
      if (.NOT.(a*vel(Y) .EQ. b*vel(X))) then
         time3=(b*(startpt(X)-node1(X))+(node1(Y)-startpt(Y))*a)/
     1          (a*vel(Y)-b*vel(X))
      endif

      t=time1
      if (((time2 .LT. 0) .AND. (time2 .GT. t)) .OR.
     1    ((time2 .GT. 0) .AND. (time2 .LT. t))) then
         t=time2
      endif
      if (((time3 .LT. 0) .AND. (time3 .GT. t)) .OR.
     1    ((time3 .GT. 0) .AND. (time3 .LT. t))) then
         t=time3
      endif
```

```
      if (t .GT. 1) then
C        Intersection point exists in the direction of the velocity
C        vector.
         endpt(X) = startpt(X) + vel(X)*t
         endpt(Y) = startpt(Y) + vel(Y)*t
         endpt(Z) = startpt(Z) + vel(Z)*t

         if (((endpt(X)-min(node1(X),node2(X))) .GT. -POINTERR) .AND.
     1       ((endpt(X)-max(node1(X),node2(X))) .LT. POINTERR) .AND.
     2       ((endpt(Y)-min(node1(Y),node2(Y))) .GT. -POINTERR) .AND.
     3       ((endpt(Y)-max(node1(Y),node2(Y))) .LT. POINTERR) .AND.
     4       ((endpt(Z)-min(node1(Z),node2(Z))) .GT. -POINTERR) .AND.
     5       ((endpt(Z)-max(node1(Z),node2(Z))) .LT. POINTERR)) then
C           Intersection point is contained within the line segment
C           between node1 and node2.
            ixedge = .TRUE.
         endif
      endif

      return
      end

C -------------------------------------------------------------------
      logical function ixside(startpt,vele,node1,node2,node3,t,endpt)

C     This function tests the plane described by the given nodes to
C     see if the given vector intersects the plane. If it does, the
C     value .TRUE. is returned, along with the time of travel and the
C     point of intersection. Otherwise, the value .FALSE. is returned
C     and other parameters remain undefined.
C -------------------------------------------------------------------
      parameter (X=1, Y=2, Z=3)

      double precision startpt(3), vele(3), node1(3), node2(3),
     1                 node3(3), endpt(3),t
      double precision a, b, c, d

      ixside = .FALSE.
      t = 0.0

      a=(node2(Z)-node1(Z))*(node3(Y)-node1(Y))-
     1  (node3(Z)-node1(Z))*(node2(Y)-node1(Y))
      b=(node3(Z)-node1(Z))*(node2(X)-node1(X))-
     1  (node2(Z)-node1(Z))*(node3(X)-node1(X))
      c=(node2(Y)-node1(Y))*(node3(X)-node1(X))-
     1  (node3(Y)-node1(Y))*(node2(X)-node1(X))
      d=-(a*node1(X)+b*node1(Y)+c*node1(Z))

      if (.NOT. ((a*vele(X) .EQ. 0) .AND. (b*vele(Y) .EQ. 0) .AND.
     1           (c*vele(Z) .EQ. 0))) then
         t= -(a*startpt(X)+b*startpt(Y)+c*startpt(Z)+d) /
     1       (a*vele(X)+b*vele(Y)+c*vele(Z))
      endif

      if (t .GT. 1) then
C        Intersection point is in the direction of the velocity vector
         endpt(X) = startpt(X) + t*vele(X)
```

```
            endpt(Y) = startpt(Y) + t*vele(Y)
            endpt(Z) = startpt(Z) + t*vele(Z)
            ixside = .TRUE.
C           Note that it is not necessary to check whether the endpoint
C           lies within the area defined by the nodes because the program
C           parts that use this routine all check for the for the lowest
C           time of travel, a long travel time indicating that the
C           intersection is outside the element boundaries. This is done
C           because of the difficulty of deciding whether a point is
C           inside an irregularly shaped area or not.
         endif

         return
         end

C ----------------------------------------------------------------------
         double precision function min(val1,val2)

C        Given two variables, this function determines which has the
C        higher value and returns that value.
C ----------------------------------------------------------------------
         double precision val1, val2

         if (val1 .GT. val2) then
            min = val2
         else
            min = val1
         endif
         return
         end

C ----------------------------------------------------------------------
         double precision function max(val1, val2)

C        Given two variables, this function determines which has the
C        higher value and returns that value.
C ----------------------------------------------------------------------
         double precision val1, val2

         if (val1 .GT. val2) then
            max = val1
         else
            max = val2
         endif
         return
         end

C ----------------------------------------------------------------------
         subroutine normaliz(vector, unitvec)

C        This procedure returns a unit vector in the direction of the
C        given vector, by dividing the vector's components by the vector's
C        magnitude.
C ----------------------------------------------------------------------
         double precision vector(3), unitvec(3)
         double precision magnitud

         parameter (X=1, Y=2, Z=3)
```

```
      magnitud = DSQRT(vector(X)*vector(X) + vector(Y)*vector(Y) +
     1                 vector(Z)*vector(Z))
      if (magnitud .NE. 0) then
         unitvec(X) = vector(X)/magnitud
         unitvec(Y) = vector(Y)/magnitud
         unitvec(Z) = vector(Z)/magnitud
      else
         unitvec(X) = 0
         unitvec(Y) = 0
         unitvec(Z) = 0
      endif
      return
      end

C -----------------------------------------------------------------------
      double precision function dot(vector1, vector2)

C     This function returns the dot product of two vectors. The if
C     statement is to get rid of annoying roundoff errors.
C -----------------------------------------------------------------------
      double precision vector1(3), vector2(3)
      double precision DOTERR, temp
      parameter (DOTERR=1D-60, X=1, Y=2, Z=3)

      dot = 0
      temp = vector1(X)*vector2(X) + vector1(Y)*vector2(Y) +
     1       vector1(Z)*vector2(Z)
      if (DABS(temp) .LT. DOTERR) then
         dot = 0.0
      else
         dot = temp
      endif
      return
      end

C -----------------------------------------------------------------------
      double precision function magnitud(vector)

C     This function returns the magnitude of a given input vector.
C -----------------------------------------------------------------------
      double precision vector(3)

      magnitud = DSQRT(vector(1)*vector(1) + vector(2)*vector(2) +
     1                 vector(3)*vector(3))
      return
      end
C -----------------------------------------------------------------------
C -----------------------------------------------------------------------
      subroutine prbanner(unit)

C     this procedure prints the heading banner for each of the pathway
C     tests in the detail file.
C -----------------------------------------------------------------------
      integer unit

      write(unit,*) '     '
      write(unit,*) '     '
```

```
      write(unit,*) '                 ----------------------------------
-----'
      write(unit,*) '                 P A T H L I N E   C O O R D I N A
T E S'
      write(unit,*) '                 ----------------------------------
-----'
      write(unit,*) ' ========= ========= ========= ======= ======
====== ============ ============'
      write(unit,*) '
Accum.      Accum.'
      write(unit,*) '    x          y          z      Element  atedge
atnode  Travel      Travel'
      write(unit,*) '                                      #
Time      Distance'
      write(unit,*) '    (m)        (m)        (m)
(s)'
      write(unit,*) ' ========= ========= ========= ======= ======
====== ============ ============'
      return
      end

C ---------------------------------------------------------------------
      subroutine setupsum(sumfile)

C     this procedure sets up the summary file and prints the files
C     associated banner.
C ---------------------------------------------------------------------
      character*30 sumfile

      open(7,FILE=sumfile)
      write(7,*)'
      ================================================:
      write(7,*)'       P A R T I C L E   T R A C K I N G   R E S U L T
S'
      write(7,*)'
      ================================================'
      write(7,*) '     '
      write(7,*)'Element      Initial location            Exit location
Travel  Status'
      write(7,*)'number       x        y        z          x        y
z        Time'
      write(7,*)'             (m)      (m)      (m)        (m)      (m)
(m)      (years)'

write(7,*)'=============================================================
================'
      write(7,*) '     '
      return
      end

C ---------------------------------------------------------------------
      subroutine rdconfig()

C     This procedure reads the input and output filenames from the
C     configuration file.
C ---------------------------------------------------------------------
      common /FILES/ elemf, velf, elemnetf, nodelstf,
     1               nodef, sumf, outf, inf
```

```
      character*30 elemf, velf, elemnetf, nodelstf,
     1             nodef, sumf, outf, inf

      character*30 configf

10    format(A30)
      write(*,*) 'Enter configuration file name: '
      read(*,10) configf

      open(10, FILE=configf)
      read(10,10) elemf
      read(10,10) velf
      read(10,10) nodelstf
      read(10,10) nodef
      read(10,10) sumf
      read(10,10) outf
      read(10,10) inf
      close(10)

      return
      end

C -------------------------------------------------------------------
      subroutine rdnodeco(nodes,nodef)

C     This procedure reads the file that contains the coordinates for
C     each of the nodes. It assumes that the records of the file are
C     ordered by the node number so that later reference to a specific
C     node can use the array index rather than sequentially searching.
C -------------------------------------------------------------------
      include "max.f"
      double precision nodes(MAXNODES, 3)
      character*30 nodef

      integer node, nodeID
      parameter (X=1,Y=2,Z=3)

1     format(I5, 3(D10.3))

      open(10, FILE=nodef)
      do 2 node=1, MAXNODES
         read(10, 1) nodeID, nodes(node,X), nodes(node,Y),
     1   nodes(node,Z)
         if (node .NE. nodeID) then
            write(*,*) 'Node number mismatch reading node file'
            write(*,*) 'Expected ',node,' Found ',nodeID
         end if
2     continue
      close(10)
      return
      end

C -------------------------------------------------------------------
      subroutine rdnolst(nodelist, nodelstf)

C     This procedure builds the list that stores the node IDs and
C     the element IDs that have that node in common.
C -------------------------------------------------------------------
```

```
      include "max.f"
      integer nodelist(MAXNODES, 0:16)
      character*30 nodelstf

      integer nodeID, i, elemID, numelem
      parameter (COUNT=0)

3     format(2(I5))
4     format(I7)

      open(10, FILE=nodelstf)
5     read(10,3,END=7) nodeID, numelem
      nodelist(nodeID,COUNT) = numelem-1

      do 6 i=1,nodelist(nodeID,COUNT)
         read(10,4) elemID
         nodelist(nodeID, i) = elemID
6     continue
      goto 5

7     continue
      close(10)
      return
      end

C -------------------------------------------------------------------
C -------------------------------------------------------------------
      subroutine corange(elem, max, min)

C     This procedure determines the largest and smallest x, y, and z
C     coordinates used for the nodes of the given element. Each node
C     of the element is compared to the present maximum and minimum
C     values and those are updated if appropriate.
C -------------------------------------------------------------------
      integer elem, side
      double precision max(3), min(3)

      parameter (NODE1=1,NODE2=2)
      parameter (X=1, Y=2, Z=3)
      double precision MAXDOUBLE

      include "max.f"
      common /ELNET/ elside(-1:MAXELEMS, 6, 5),
     1               elused(-1:MAXELEMS,6,4),
     2               eledge(-1:MAXELEMS, 12, 2),
     3               elvel(-1:MAXELEMS, 3)
      common /NODES/ nodes(MAXNODES, 3)
      integer elside, elused, eledge
      double precision nodes, elvel

      integer nod1id, nod2id

      MAXDOUBLE=1.7D+308
      min(X)=MAXDOUBLE
      min(Y)=MAXDOUBLE
      min(Z)=MAXDOUBLE
      max(X)=-MAXDOUBLE
      max(Y)=-MAXDOUBLE
```

```
      max(Z)=-MAXDOUBLE

      do 20 side=1,6
        nodlid = elside(elem, side, NODE1)
        nod2id = elside(elem, side, NODE2)
        if (nodes(nodlid,X) .GT. max(X)) then
            max(X) = nodes(nodlid,X)
        endif
        if (nodes(nod2id,X) .GT. max(X)) then
           max(X) = nodes(nod2id,X)
        endif
        if (nodes(nodlid,X) .LT. min(X)) then
            min(X) = nodes(nodlid,X)
        endif
        if (nodes(nod2id,X) .LT. min(X)) then
           min(X) = nodes(nod2id,X)
        endif
        if (nodes(nodlid,Y) .GT. max(Y)) then
            max(Y) = nodes(nodlid,Y)
        endif
        if (nodes(nod2id,Y) .GT. max(Y)) then
           max(Y) = nodes(nod2id,Y)
        endif
        if (nodes(nodlid,Y) .LT. min(Y)) then
            min(Y) = nodes(nodlid,Y)
        endif
        if (nodes(nod2id,Y) .LT. min(Y)) then
           min(Y) = nodes(nod2id,Y)
        endif
        if (nodes(nodlid,Z) .GT. max(Z)) then
            max(Z) = nodes(nodlid,Z)
        endif
        if (nodes(nod2id,Z) .GT. max(Z)) then
           max(Z) = nodes(nod2id,Z)
        endif
        if (nodes(nodlid,Z) .LT. min(Z)) then
            min(Z) = nodes(nodlid,Z)
        endif
        if (nodes(nod2id,Z) .LT. min(Z)) then
           min(Z) = nodes(nod2id,Z)
        endif
20    continue

      return
      end

C -----------------------------------------------------------------------
      logical function getelem(nodelid, node2id, startpt, oldvel,
     1        travtime, exitpt, exitside,
     2        elemid, nextelid, largemag, prevelem)

C     This function will find which of the elements that surround
C     the edge we are presently on provides the best route for further
C     motion. If a suitable element is found (the decision being based
C     on the largest dot product among those with positive times of
C     travel), then the particle is advanced and the end point of the
C     path, the velocity of travel and the travel time are returned.
C     If no suitable side is found, the function returns the value
```

```
C     .FALSE..
C -------------------------------------------------------------------
      integer nodelid,node2id,exitside,elemid,nextelid,prevelem
      double precision startpt(3),oldvel(3),travtime,
     1   exitpt(3),largemag

      external thruelem, dot, magnitud
      logical thruelem
      double precision dot, magnitud

      include "max.f"
      common /ELNET/ elside(-1:MAXELEMS, 6, 5),
     1               elused(-1:MAXELEMS,6,4),
     2               eledge(-1:MAXELEMS, 12, 2),
     3               elvel(-1:MAXELEMS, 3)
      common /LIST/  nolstarr(MAXNODES, 0:16)
      double precision elvel
      integer elside, elused, eledge, nolstarr

      integer curl, cur2, elem, nextelem, endside
      double precision elemvel(3), t, endpt(3)
      parameter (X=1,Y=2,Z=3,COUNT=0)

      largemag = 0.0
      getelem = .FALSE.
      do 20 curl=1,nolstarr(nodelid, COUNT)
         do 20 cur2=1,nolstarr(node2id, COUNT)
            if (nolstarr(nodelid,curl) .EQ.
     1          nolstarr(node2id,cur2)) then
C              This is an element we have to check.
               elem = nolstarr(nodelid, curl)
               do 10 i=1,3
                  elemvel(i) = elvel(elem, i)
10             continue
               if (thruelem(startpt,elem,endpt,endside,nextelem,
     1                     t, prevelem) .AND.
     2              (dot(oldvel,elemvel) .GE. 0) .AND.
     3              (magnitud(elemvel) .GT. largemag)) then
C                 Element chosen is best so far - move through it
                  exitpt(X) = endpt(X)
                  exitpt(Y) = endpt(Y)
                  exitpt(Z) = endpt(Z)
                  largemag = magnitud(elemvel)
                  exitside = endside
                  nextelid = nextelem
                  elemid = elem
                  travtime = t
                  getelem = .TRUE.
               endif
            endif
20       continue

         return
         end

C -------------------------------------------------------------------
      logical function elemfr(node,startpt,oldvel,elemid,
     1   nextelem,travtime,exitpt,exitside,largemag,prevelem)
```

```
C     When at a node, this function finds which of the surrounding
C     elements is most favourable for moving into, based on a
C     criteria of the largest dot product while still having a positive
C     travel time. If no suitable element is found, the result is
C     returned as .FALSE.. Otherwise, the data for the move are
C     returned.
C ---------------------------------------------------------------------
      integer node, elemid, nextelem, exitside, prevelem
      double precision startpt(3), oldvel(3),
     1   travtime, exitpt(3), largemag

      include "max.f"
      common /ELNET/ elside(-1:MAXELEMS, 6, 5),
     1               elused(-1:MAXELEMS,6,4),
     2               eledge(-1:MAXELEMS, 12, 2),
     3               elvel(-1:MAXELEMS, 3)
      common /LIST/  nolstarr(MAXNODES, 0:16)
      double precision elvel
      integer elside, elused, eledge, nolstarr

      external thruelem, dot, magnitud
      logical thruelem
      double precision dot, magnitud

      parameter (COUNT=0,X=1,Y=2,Z=3)
      integer current, elem, nextel, i, endside
      double precision elemvel(3), t, endpt(3)

      largemag = 0.0
      elemfr = .FALSE.

      do 40 current=1,nolstarr(node,COUNT)
         elem = nolstarr(node,current)
         do 30 i=1,3
            elemvel(i) = elvel(elem,i)
30       continue
         if (thruelem(startpt,elem,endpt,endside,nextel,t,prevelem)
     1        .AND. (dot(oldvel,elemvel) .GE. 0)
     2        .AND. (magnitud(elemvel) .GT. largemag)) then
C           This is the best element so far.
            travtime = t
            elemid = elem
            exitpt(X) = endpt(X)
            exitpt(Y) = endpt(Y)
            exitpt(Z) = endpt(Z)
            exitside = endside
            nextelem = nextel
            largemag = magnitud(elemvel)
            elemfr = .TRUE.
         endif
40    continue

      return
      end

C ---------------------------------------------------------------------
      logical function getside(elemid,edge,lastvel,startpt,
```

```
     1   travtime,chiefel,endedge,exitpt,velocity,largemag,exside)

C     This function will find which of the possible sides available
C     has the most favourable velocity. If a suitable side is found,
C     the particle is advanced and the end point of the path, the
C     velocity of travel and the travel time are returned. If no suit-
C     able side is found, the value .FALSE. is returned and other
C     values are undefined. Not that it is not possible for the side
C     the particle just came from to be the chosen side as the time
C     returned for that motion would be zero.
C ------------------------------------------------------------------
      integer elemid, edge, chiefel, endedge, exside
      double precision lastvel(3), startpt(3),
     1   travtime, exitpt(3), velocity(3), largemag

      include "max.f"
      common /ELNET/ elside(-1:MAXELEMS, 6, 5),
     1               elused(-1:MAXELEMS,6,4),
     2               eledge(-1:MAXELEMS, 12, 2),
     3               elvel(-1:MAXELEMS, 3)
      common /LIST/  nolstarr(MAXNODES, 0:16)
      double precision elvel
      integer elside, elused, eledge, nolstarr

      external alongsid, magnitud, dot
      logical alongsid
      double precision magnitud, dot
      integer side, cur1, cur2, exitedge, nodelid, node2id, adelem
      double precision t, endpt(3), vel(3)
      parameter (NODE1=1,NODE2=2,NODE3=3,NODE4=4,COUNT=0)
      parameter (X=1,Y=2,Z=3)

      travtime = 0.0
      largemag = 0.0
      getside = .FALSE.

      nodelid = eledge(elemid, edge, NODE1)
      node2id = eledge(elemid, edge, NODE2)

      do 20 cur1=1,nolstarr(nodelid, COUNT)
         do 20 cur2=1,nolstarr(node2id, COUNT)
            if ((nolstarr(nodelid,cur1) .EQ. nolstarr(node2id,cur2))
     1        .AND.(.NOT.(nolstarr(nodelid,cur1) .EQ. elemid))) then
C              This element bears potential for movement.
               adelem = nolstarr(nodelid,cur1)
               do 10 side=1,6
                  if (((elside(adelem,side,NODE1)
     1                  .EQ. eledge(elemid,edge,NODE1)) .OR.
     2                 (elside(adelem,side,NODE2)
     3                  .EQ. eledge(elemid,edge,NODE1)) .OR.
     4                 (elside(adelem,side,NODE3)
     5                  .EQ. eledge(elemid,edge,NODE1)) .OR.
     6                 (elside(adelem,side,NODE4)
     7                  .EQ. eledge(elemid,edge,NODE1))) .AND.
     8                ((elside(adelem,side,NODE1)
     9                  .EQ. eledge(elemid,edge,NODE2)) .OR.
     1                 (elside(adelem,side,NODE2)
     2                  .EQ. eledge(elemid,edge,NODE2)) .OR.
```

```
     3                 (elside(adelem,side,NODE3)
     4                  .EQ. eledge(elemid,edge,NODE2)) .OR.
     5                 (elside(adelem,side,NODE4)
     6                  .EQ. eledge(elemid,edge,NODE2)))) then
C                    The nodes of this side are the same as the
C                    nodes in the start edge so this is a
C                    potential side for moving across.
                     if (alongsid(startpt,adelem,side,endpt,
     1                            vel,exitedge,t) .AND.
     2                   (dot(lastvel,vel) .GE. 0) .AND.
     3                   (t .GT. 1) .AND.
     4                   (magnitud(vel) .GT. largemag)) then
C                        Best side so far.
                         getside = .TRUE.
                         travtime = t
                         endedge = exitedge
                         exitpt(X) = endpt(X)
                         exitpt(Y) = endpt(Y)
                         exitpt(Z) = endpt(Z)
                         velocity(X) = vel(X)
                         velocity(Y) = vel(Y)
                         velocity(Z) = vel(Z)
                         exside = side
                         largemag = magnitud(vel)
C                        The element the particle is said to be 'in'
C                        at this time.
                         chiefel = adelem
                     endif
                 endif
10             continue
           endif
20    continue
      return
      end

C -------------------------------------------------------------------
      logical function getedge(node,lastvel,chiefel,travtime,
     1   endpt,vel,exitnode,largemag,endedge)

C     In the situation where the particle is stuck at a node, this
C     function checks each of the edges that the particle might be
C     able to move along. The edges to be checked are determined by
C     testing each edge of each of the elements that are identified
C     as possible elements for moving into. This checks some edges
C     more than once but the process is quick and little time should
C     be lost in the use of this simple algorithm. The best edge for
C     the next move is chosen on the basis of the one with the velocity
C     that yields the highest dot product with the last known velocity.
C     As in other routines, however, moves with t=0 are not allowed as
C     those would leave the particle where it is, forcing an endless
C     loop.
C -------------------------------------------------------------------
      integer node, chiefel, exitnode, endedge
      double precision lastvel(3), travtime, endpt(3), vel(3), largemag

      include "max.f"
      common /ELNET/ elside(-1:MAXELEMS, 6, 5),
     1               elused(-1:MAXELEMS,6,4),
```

```
     2               eledge(-1:MAXELEMS, 12, 2),
     3               elvel(-1:MAXELEMS, 3)
      common /NODES/ nodes(MAXNODES, 3)
      common /LIST/  nolstarr(MAXNODES, 0:16)
      double precision elvel
      integer elside, elused, eledge, nolstarr
      double precision nodes

      external alongedg, dot, magnitud
      double precision dot, magnitud
      parameter (NODE1=1,NODE2=2,COUNT=0,X=1,Y=2,Z=3)
      integer edge, current, i, elem
      double precision exitpt(3), exitvel(3), t, nodecord(3), POINTERR
      parameter (POINTERR=1D-03)

      getedge = .FALSE.
      largemag = 0.0

      do 30 current=1,nolstarr(node,COUNT)
C        Don't check non-existent elements.
         if (nolstarr(node,current) .GT. 0) then
            elem = nolstarr(node,current)
            do 20 edge=1,12
               if ((eledge(elem,edge,NODE1) .EQ. node) .OR.
     1              (eledge(elem,edge,NODE2) .EQ. node)) then
                  do 10 i=1,3
                     nodecord(i) = nodes(node,i)
10                continue
                  call alongedg(edge,elem,nodecord,exitpt,exitvel,t)
                  if ((dot(lastvel,exitvel) .GE. 0) .AND.
     1                 (magnitud(exitvel) .GT. largemag)) then
C                    This is a valid edge for travel and is the
C                    best so far.
                     largemag = magnitud(exitvel)
                     chiefel = elem
                     endpt(X) = exitpt(X)
                     endpt(Y) = exitpt(Y)
                     endpt(Z) = exitpt(Z)
                     endedge = edge
                     vel(X) = exitvel(X)
                     vel(Y) = exitvel(Y)
                     vel(Z) = exitvel(Z)
                     travtime = t
                     getedge = .TRUE.

C                    Record which node the particle is presently at.
                     if ((DABS(nodes(eledge(elem,edge,NODE2),X)
     1                  -exitpt(X)) .LT. POINTERR) .AND.
     2                   (DABS(nodes(eledge(elem,edge,NODE2),Y)
     3                  -exitpt(Y)) .LT. POINTERR) .AND.
     4                   (DABS(nodes(eledge(elem,edge,NODE2),Z)
     5                  -exitpt(Z)) .LT. POINTERR)) then
                        exitnode = eledge(elem, edge, NODE2)
                     else
                        exitnode = eledge(elem, edge, NODE1)
                     endif

                  endif
```

```
               endif

20          continue
         endif

30    continue
      return
      end

C -------------------------------------------------------------------
      logical function sidefr(node,lastvel,travtime,chiefel,
     1   exitedge,exitpt,vel,largemag,exside)

C     Checks each of the sides the particle might travel along to find
C     which, if any, has the velocity with the largest positive dot
C     product with the last velocity. Very similar to the routine
C     getside, but this one must check four elements in a different
C     arrangement, as the particle is residing at a node rather than
C     along an edge. If a suitable side is found, the data for travel-
C     ling across it are returned.
C -------------------------------------------------------------------
      integer node, chiefel, exitedge, exside
      double precision lastvel(3), travtime, exitpt(3),
     1                 vel(3), largemag


      include "max.f"
      common /ELNET/ elside(-1:MAXELEMS, 6, 5),
     1               elused(-1:MAXELEMS,6,4),
     2               eledge(-1:MAXELEMS, 12, 2),
     3               elvel(-1:MAXELEMS, 3)
      common /NODES/ nodes(MAXNODES, 3)
      common /LIST/  nolstarr(MAXNODES, 0:16)
      double precision elvel
      integer elside, elused, eledge, nolstarr
      double precision nodes

      external alongsid, dot, magnitud
      logical alongsid
      double precision dot, magnitud
      integer side, endedge, current, i, elem
      double precision t, endpt(3), planvel(3), nodecord(3)
      parameter (COUNT=0,X=1,Y=2,Z=3)
      parameter (NODE1=1,NODE2=2,NODE3=3,NODE4=4)

      sidefr = .FALSE.
      travtime = 0.0
      largemag = 0.0

      do 20 current=1,nolstarr(node,COUNT)
         elem = nolstarr(node, current)

         do 10 side=1,6
            if ((elside(elem,side,NODE1) .EQ. node) .OR.
     1          (elside(elem,side,NODE2) .EQ. node) .OR.
     2          (elside(elem,side,NODE3) .EQ. node) .OR.
     3          (elside(elem,side,NODE4) .EQ. node)) then
C              This is one of the sides we have to check.
```

```
                do 5 i=1,3
                   nodecord(i) = nodes(node,i)
5               continue
                if (alongsid(nodecord,elem,side,endpt,planvel,endedge,t)
     1             .AND. (dot(planvel,lastvel) .GE. 0)
     2             .AND. (magnitud(planvel) .GT. largemag)) then
C                  Best side found so far.
                   sidefr = .TRUE.
                   largemag = magnitud(planvel)
                   chiefel = elem
                   travtime = t
                   exitedge = endedge
                   exitpt(X) = endpt(X)
                   exitpt(Y) = endpt(Y)
                   exitpt(Z) = endpt(Z)
                   exside = side
                   vel(X) = planvel(X)
                   vel(Y) = planvel(Y)
                   vel(Z) = planvel(Z)
                endif
             endif
10        continue
20     continue

       return
       end

C ---------------------------------------------------------------------
       subroutine findvel(pt1, pt2, pt3, vel1, vel2, planevel)

C      A subroutine to find the direction and magnitude of the velocity
C      field in the plane between two elements. Given three points in
C      the plane and the velocity fields in the two elements the
C      routine finds the components of the velocities in the plane and
C      adds them to get the direction of the resultant velocity. The
C      magnitude is taken to be the greater of the components of the
C      components of the element fields along the calculated direction.
C      This will be used in situations where the velocity fields in two
C      elements are in opposition, forcing travel along their common
C      boundary plane.
C ---------------------------------------------------------------------
       double precision pt1(3), pt2(3), pt3(3),
     1   vel1(3), vel2(3), planevel(3)
       parameter (X=1, Y=2, Z=3)

       external normaliz, dot, max
       double precision dot, max

       double precision perp(3), unitperp(3)
       double precision proj1(3), proj2(3)
       double precision tempfct
       double precision direct(3), unitdir(3)
       double precision magv1cmp, magv2cmp, mag, a, b, c

       a = (pt2(Z)-pt1(Z))*(pt3(Y)-pt1(Y))-
     1     (pt3(Z)-pt1(Z))*(pt2(Y)-pt1(Y))
       b = (pt3(Z)-pt1(Z))*(pt2(X)-pt1(X))-
     1     (pt2(Z)-pt1(Z))*(pt3(X)-pt1(X))
```

```
      c = (pt2(Y)-pt1(Y))*(pt3(X)-pt1(X))-
     1    (pt3(Y)-pt1(Y))*(pt2(X)-pt1(X))

      if (.NOT. ((a .EQ. 0) .AND. (b .EQ. 0) .AND. (c .EQ. 0))) then
         perp(X) = a
         perp(Y) = b
         perp(Z) = c
         call normaliz(perp, unitperp)

         tempfct = dot(unitperp, vel1)
         proj1(X) = vel1(X) - tempfct*unitperp(X)
         proj1(Y) = vel1(Y) - tempfct*unitperp(Y)
         proj1(Z) = vel1(Z) - tempfct*unitperp(Z)

         tempfct = dot(unitperp, vel2)
         proj2(X) = vel2(X) - tempfct*unitperp(X)
         proj2(Y) = vel2(Y) - tempfct*unitperp(Y)
         proj2(Z) = vel2(Z) - tempfct*unitperp(Z)
      else
         proj1(X) = vel1(X)
         proj1(Y) = vel1(Y)
         proj1(Z) = vel1(Z)

         proj2(X) = vel2(X)
         proj2(Y) = vel2(Y)
         proj2(Z) = vel2(Z)
      endif

      direct(X) = proj1(X) + proj2(X)
      direct(Y) = proj1(Y) + proj2(Y)
      direct(Z) = proj1(Z) + proj2(Z)
      call normaliz(direct, unitdir)
      magv1cmp = dot(unitdir, vel1)
      magv2cmp = dot(unitdir, vel2)
      mag = max(magv1cmp, magv2cmp)
      planevel(X) = mag*unitdir(X)
      planevel(Y) = mag*unitdir(Y)
      planevel(Z) = mag*unitdir(Z)

      return
      end

C -----------------------------------------------------------------------
      subroutine resvel(node1, node2, velmag)

C     This procedure finds the velocity along an edge by summing the
C     projected velocity components of the surrounding elements. The
C     projected components are found by taking the dot product of the
C     velocity and the edge vectors. The magnitude of the resulting
C     velocity is returned (the direction is obviously alng the edge).
C     Positive or negative signifies approaching or retreating from
C     the second node.
C -----------------------------------------------------------------------
      integer node1, node2
      double precision velmag

      include "max.f"
      common /ELNET/ elside(-1:MAXELEMS, 6, 5),
```

```
     1               elused(-1:MAXELEMS,6,4),
     2               eledge(-1:MAXELEMS, 12, 2),
     3               elvel(-1:MAXELEMS, 3)
      common /NODES/ nodes(MAXNODES, 3)
      common /LIST/  nolstarr(MAXNODES, 0:16)
      integer elside, elused, eledge, nolstarr
      double precision elvel, nodes

      parameter (X=1, Y=2, Z=3, COUNT=0)
      external normaliz, dot
      double precision edgevect(3), unitvect(3), point1(3), point2(3),
     1                 vel1(3), dot
      integer i, j

      do 20 i=1,3
         point1(i) = nodes(node1, i)
         point2(i) = nodes(node2, i)
20    continue

      edgevect(X) = point2(X) - point1(X)
      edgevect(Y) = point2(Y) - point1(Y)
      edgevect(Z) = point2(Z) - point1(Z)
      call normaliz(edgevect, unitvect)

      velmag = 0.0
      do 30 i=1,nolstarr(node1, COUNT)
         do 30 j=1,nolstarr(node2, COUNT)
            if (nolstarr(node1,i) .EQ. nolstarr(node2,j)) then
               vel1(X) = elvel(nolstarr(node1,i),X)
               vel1(Y) = elvel(nolstarr(node1,i),Y)
               vel1(Z) = elvel(nolstarr(node1,i),Z)
               if (dot(vel1, unitvect) .GT. velmag) then
                  velmag = dot(vel1, unitvect)
               endif
            endif
30    continue
      end

C -----------------------------------------------------------------------
      subroutine readinit(more, elemid, position, vperiod, infunit)

C     This procedure reads the initial coordinates for pathway
C     analysis from the user generated input file <filename>.in
C -----------------------------------------------------------------------
      integer elemid, infunit
      logical more
      double precision position(3), vperiod
      parameter (X=1,Y=2,Z=3)

10    format(I7, D8.3, 2(D10.3))
      more = .FALSE.
      read(infunit,10,END=20) elemid,position(X),position(Y),position(Z)
      more = .TRUE.
20    vperiod = 1.0D20
      return
      end

C -----------------------------------------------------------------------
```

```
      logical function thruelem(startpt, elemid, exitpt, exitside,
     1                          nextelem, travtime, prevelem)

C     This function takes an elementid and a start point and finds
C     the exit point from that element according to the velocity
C     vector in that element. If a valid intersection is found, the
C     time of travel, the exit side and the exit point will be
C     returned. If the position of the start point given the pre-
C     vailing velocity vector is such that no motion is possible,
C     the function will return a zero travel time and other output
C     will be undefined.
C ---------------------------------------------------------------------
      integer elemid, exitside, nextelem, prevelem
      double precision startpt(3), exitpt(3), travtime
      external ixside
      logical ixside

      parameter (POINTERR=1D-3,NODE1=1,NODE2=2,NODE3=3,X=1,Y=2,Z=3)

      include "max.f"
      common /ELNET/ elside(-1:MAXELEMS, 6, 5),
     1               elused(-1:MAXELEMS,6,4),
     2               eledge(-1:MAXELEMS, 12, 2),
     3               elvel(-1:MAXELEMS, 3)
      common /NODES/ nodes(MAXNODES, 3)
      integer elside, elused, eledge
      double precision elvel, nodes

      integer side
      logical id, onside, result
      double precision endpt(3), max(3), min(3), t
      double precision nod1cord(3), nod2cord(3), nod3cord(3),
     1                 velcord(3)

      id = .FALSE.
      onside = .FALSE.
C     Assume we can't move into this element
      thruelem = .FALSE.
C     Initialize travel time so that if no intercept is found
C     then the time of travel can be used a flag.
      travtime = 0

C     Test all sides until right one is found
      do 100 side=1,6
c
c        This bit may seem clumsy, but it's a result of my more
c        or less 'literal' translation of the pascal program. The
c        problem is that you can't pass a partially subscripted
c        array as an argument in FORTRAN. Take a look at what
c        MoveThroughElement does.
c
         do 20 i=1,3
            nod1cord(i)=nodes(elside(elemid,side,NODE1), i)
            nod2cord(i)=nodes(elside(elemid,side,NODE2), i)
            nod3cord(i)=nodes(elside(elemid,side,NODE3), i)
            velcord(i)=elvel(elemid, i)
20       continue
```

```
      result = ixside(startpt,velcord,nod1cord,nod2cord,nod3cord,
     1       t,endpt)
         if (result .AND. ((t .LT. travtime) .OR.
     1      (travtime .EQ. 0))) then
C           This is the best side for exit so far.
            call corange(elemid, max, min)
            if (((endpt(X)-min(X)) .GT. -POINTERR) .AND.
     1         ((endpt(Y)-min(Y)) .GT. -POINTERR) .AND.
     2         ((endpt(Z)-min(Z)) .GT. -POINTERR) .AND.
     3         ((endpt(X)-max(X)) .LT. POINTERR) .AND.
     4         ((endpt(Y)-max(Y)) .LT. POINTERR) .AND.
     5         ((endpt(Z)-max(Z)) .LT. POINTERR) .AND.
     6         (prevelem .NE. elside(elemid, side, 5))) then
C              Exit point is actually in the element.
               exitpt(X) = endpt(X)
               exitpt(Y) = endpt(Y)
               exitpt(Z) = endpt(Z)
               exitside = side
               nextelem = elside(elemid, side, 5)
               travtime = t
               id = .TRUE.
C              There is a valid move to be made.
               thruelem = .TRUE.
            endif
         else
            call corange(elemid, max, min)
            endpt(X) = startpt(X)
            endpt(Y) = startpt(Y)
            endpt(Z) = startpt(Z)
            if (((endpt(X)-min(X)) .GT. -POINTERR) .AND.
     1         ((endpt(Y)-min(Y)) .GT. -POINTERR) .AND.
     2         ((endpt(Z)-min(Z)) .GT. -POINTERR) .AND.
     3         ((endpt(X)-max(X)) .LT. POINTERR) .AND.
     4         ((endpt(Y)-max(Y)) .LT. POINTERR) .AND.
     5         ((endpt(Z)-max(Z)) .LT. POINTERR) .AND.
     6         (t .EQ. 0) .AND. (.NOT. id)) then
               onside = .TRUE.
               exitside = side
               nextelem = elside(elemid, side, 5)
            endif
         endif

100   continue

      if ((.NOT. id) .AND. onside) then
         t = 0
      endif

      return
      end

C -----------------------------------------------------------------------
      logical function alongsid(startpt, elemlid, side, exitpt, vele,
     1                    exitedge, travtime)

C     When travel through the next element is impossible, generally
C     due to opposing velocity fields in the two elements, the part-
C     icle travels along the side instead. The velocity vector
```

```
C     resulting from the combination of the velocity fields of the
C     two elements is used to determine the exit point. Each edge
C     of the side is tested to see if that is where the velocity
C     vector intercepts. The exit point, time of travel, velocity
C     travelled at and edge exitted on are returned.
C ---------------------------------------------------------------------
      double precision startpt(3), exitpt(3), vele(3), travtime
      integer elemlid, side, exitedge

      include "max.f"
      common /ELNET/ elside(-1:MAXELEMS, 6, 5),
     1               elused(-1:MAXELEMS,6,4),
     2               eledge(-1:MAXELEMS, 12, 2),
     3               elvel(-1:MAXELEMS, 3)
      common /NODES/ nodes(MAXNODES, 3)
      integer elside, elused, eledge
      double precision elvel, nodes

      external findvel, ixedge
      logical ixedge
      integer i, edge
      double precision t, endpt(3), netvel(3), resvel(3)
      double precision nod1cord(3), nod2cord(3), nod3cord(3),
     1                 vel1(3), vel2(3)
      parameter (X=1, Y=2, Z=3, NODE1=1, NODE2=2, NODE3=3, ID=5)

      alongsid = .FALSE.
      travtime = 0.0

C     This looks different than the Pascal version, but it's to
C     get around the fact that you can't pass partially sub-
C     scripted arrays in FORTRAN.

      do 20 i=1,3
         nod1cord(i)=nodes(elside(elemlid,side,NODE1),i)
         nod2cord(i)=nodes(elside(elemlid,side,NODE2),i)
         nod3cord(i)=nodes(elside(elemlid,side,NODE3),i)
         vel1(i)=elvel(elemlid,i)
         vel2(i)=elvel(elside(elemlid,side,ID),i)
20    continue
      call findvel(nod1cord, nod2cord, nod3cord, vel1, vel2, resvel)
      netvel(X) = resvel(X)
      netvel(Y) = resvel(Y)
      netvel(Z) = resvel(Z)

      do 30 edge=1,4
C     Four edges on each side
         do 25 i=1,3
            nod1cord(i)=nodes(eledge(elemliD,elused(elemlid,side,edge),
     1                           NODE1), i)
            nod2cord(i)=nodes(eledge(elemliD,elused(elemlid,side,edge),
     1                           NODE2), i)
25       continue

         if (ixedge(startpt,netvel,nod1cord,nod2cord,t,endpt) .AND.
     1       ((t .LT. travtime) .OR. (travtime .EQ. 0))) then
            exitpt(X) = endpt(X)
            exitpt(Y) = endpt(Y)
```

```
            exitpt(Z) = endpt(Z)
            exitedge = elused(elem1id, side, edge)
            travtime = t
            vele(X) = resvel(X)
            vele(Y) = resvel(Y)
            vele(Z) = resvel(Z)
            alongsid = .TRUE.
         endif
30    continue

      return
      end

C -----------------------------------------------------------------------
      subroutine alongedg(curredge, elemid, startpt, exitpt,
     1                    vel, travtime)

C     Finds the time necessary to move along the edge to the next
C     node. The velocity along the edge is taken to be the sum of
C     the components along the edge of the velocity vectors of the
C     surrounding three or four elements. The time of travel and
C     the destination point will be returned along with the velocity.
C -----------------------------------------------------------------------
      integer curredge, elemid
      double precision startpt(3), exitpt(3), vel(3), travtime

      include "max.f"
      common /ELNET/ elside(-1:MAXELEMS, 6, 5),
     1               elused(-1:MAXELEMS,6,4),
     2               eledge(-1:MAXELEMS, 12, 2),
     3               elvel(-1:MAXELEMS, 3)
      common /NODES/ nodes(MAXNODES, 3)
      integer elside, elused, eledge
      double precision elvel, nodes

      parameter (X=1, Y=2, Z=3, NODE1=1, NODE2=2)
      external resvel, normaliz, magnitud
      double precision distance(3), unitdir(3), point1(3), point2(3),
     1                 magvel, magnitud
      integer node1id, node2id, i

      node1id = eledge(elemid, curredge, NODE1)
      node2id = eledge(elemid, curredge, NODE2)

      call resvel(node1id, node2id, magvel)
C     Note that this velocity is just a magnitude, direction is obvious
      do 20 i=1,3
         point1(i) = nodes(node1id,i)
         point2(i) = nodes(node2id,i)
20    continue

      if (magvel .LT. 0) then
         distance(X) = point1(X) - startpt(X)
         distance(Y) = point1(Y) - startpt(Y)
         distance(Z) = point1(Z) - startpt(Z)
         exitpt(X) = point1(X)
         exitpt(Y) = point1(Y)
         exitpt(Z) = point1(Z)
```

```
      else
         distance(X) = point2(X) - startpt(X)
         distance(Y) = point2(Y) - startpt(Y)
         distance(Z) = point2(Z) - startpt(Z)
         exitpt(X) = point2(X)
         exitpt(Y) = point2(Y)
         exitpt(Z) = point2(Z)
      endif

      if (magvel .EQ. 0) then
C        Forces this to be a prohibited result - can't move.
         travtime = 0
      else
         travtime = DABS(magnitud(distance)/magvel)
      endif

      call normaliz(distance, unitdir)
      vel(X) = magvel * unitdir(X)
      vel(Y) = magvel * unitdir(Y)
      vel(Z) = magvel * unitdir(Z)

      return
      end

C ---------------------------------------------------------------------
      logical function nodebnd(currnode, elemid)

C     This function tests to see if the node in question is on the
C     boundary of the defined set of elements. If any side of the
C     element that contains the present node has an adjacent element
C     with id zero, then the node is on the boundary. (Id zero
C     corresponds to an element beyond the borders of the simulation
C     area.
C ---------------------------------------------------------------------
      integer currnode, elemid

      include "max.f"
      common /ELNET/ elside(-1:MAXELEMS, 6, 5),
     1               elused(-1:MAXELEMS,6,4),
     2               eledge(-1:MAXELEMS, 12, 2),
     3               elvel(-1:MAXELEMS, 3)
      double precision elvel
      integer elside, elused, eledge

      integer side
      logical done
      parameter (NODE1=1, NODE2=2, NODE3=3, NODE4=4, ID=5)

      done = .FALSE.
      do 20 side=1,6
         if ((elside(elemid,side,NODE1) .EQ. currnode) .OR.
     1       (elside(elemid,side,NODE2) .EQ. currnode) .OR.
     2       (elside(elemid,side,NODE3) .EQ. currnode) .OR.
     3       (elside(elemid,side,NODE4) .EQ. currnode)) then
C           Indicates that the given node is part of this side.
            if (elside(elemid,side,ID) .LT. 1) then
               done = .TRUE.
               go to 30
```

```
            endif
          endif
20      continue

30      nodebnd = done

        return
        end
C -------------------------------------------------------------------
        logical function edgebnd(curredge,nodelst1,nodelst2,elemid)

C       This function determines if the edge in question lies along the
C       boundary edge by looking at each side of each surrounding
C       element that shares the edge. If any of those sides are outside
C       sides then this must be an outside side.
C -------------------------------------------------------------------
        integer curredge, nodelst1(0:16), nodelst2(0:16), elemid

        include "max.f"
        common /ELNET/ elside(-1:MAXELEMS, 6, 5),
     1                 elused(-1:MAXELEMS,6,4),
     2                 eledge(-1:MAXELEMS, 12, 2),
     3                 elvel(-1:MAXELEMS, 3)
        double precision elvel
        integer elside, elused, eledge

        integer side, curr1, curr2, adelem
        parameter (NODE1=1,NODE2=2,NODE3=3,NODE4=4,COUNT=0,ID=5)

        edgebnd = .FALSE.
        do 40 curr1=1,nodelst1(COUNT)
           do 30 curr2=1,nodelst2(COUNT)
              if (nodelst1(curr1) .EQ. nodelst2(curr2)) then
C                This element is one of the ones on this side.
                 adelem = nodelst1(curr1)
                 do 20 side=1,5
                    if ((elside(adelem,side,NODE1) .EQ.
     1                    eledge(elemid,curredge,NODE1)) .OR.
     2                  (elside(adelem,side,NODE2) .EQ.
     3                    eledge(elemid,curredge,NODE1)) .OR.
     4                  (elside(adelem,side,NODE3) .EQ.
     5                    eledge(elemid,curredge,NODE1)) .OR.
     6                  (elside(adelem,side,NODE4) .EQ.
     7                    eledge(elemid,curredge,NODE1)) .OR.
     8                  (elside(adelem,side,NODE1) .EQ.
     9                    eledge(elemid,curredge,NODE2)) .OR.
     1                  (elside(adelem,side,NODE2) .EQ.
     2                    eledge(elemid,curredge,NODE2)) .OR.
     3                  (elside(adelem,side,NODE3) .EQ.
     4                    eledge(elemid,curredge,NODE2)) .OR.
     5                  (elside(adelem,side,NODE4) .EQ.
     6                    eledge(elemid,curredge,NODE2))) then
                       if (elside(adelem,side,ID) .LT. 1) then
                          edgebnd = .TRUE.
C                         Might as well jump out - only takes one
                          go to 30
                       endif
                    endif
```

```
20            continue
            endif
30        continue
40     continue

       return
       end

C ---------------------------------------------------------------------
       logical function chknode(point, elem, nodid)

C      This function will find whether a point is identical to any of
C      the nodes of the given element.
C ---------------------------------------------------------------------
       integer elem, nodid
       double precision point(3)

       include "max.f"
       common /ELEMLST/ elemvert(MAXELEMS,8), elemvel(MAXELEMS,3)
       common /NODES/ nodes(MAXNODES, 3)

       integer elemvert
       double precision elemvel
       double precision nodes

       integer i
       parameter(X=1,Y=2,Z=3)
       double precision POINTERR

       POINTERR = 1D-03
       chknode = .FALSE.
       do 20 i=1,8
          if ((DABS(point(X)-nodes(elemvert(elem,i),X)) .LT. POINTERR)
     1   .AND.(DABS(point(Y)-nodes(elemvert(elem,i),Y)) .LT. POINTERR)
     2   .AND.(DABS(point(Z)-nodes(elemvert(elem,i),Z)) .LT. POINTERR))
     3    then
             chknode = .TRUE.
             nodid = elemvert(elem,i)
          endif
20     continue

       return
       end

C ---------------------------------------------------------------------
       logical function chkedge(point, elem, edge)

C      This function will find if the point given is on one of the
C      edges of the element specified.
C ---------------------------------------------------------------------
       integer elem, edge
       double precision point(3)

       include "max.f"
       common /ELNET/ elside(-1:MAXELEMS, 6, 5),
      1               elused(-1:MAXELEMS,6,4),
      2               eledge(-1:MAXELEMS, 12, 2),
      3               elvel(-1:MAXELEMS, 3)
```

```
      common /NODES/ nodes(MAXNODES, 3)
      double precision elvel
      integer elside, elused, eledge
      double precision nodes

      external normaliz, dot
      double precision normaliz, dot
      integer i, node1id, node2id
      double precision vect1(3),vect2(3),uvect1(3),uvect2(3),CAEERR
      parameter (NODE1=1, NODE2=2, X=1, Y=2, Z=3)

      CAEERR = 1D-03
      chkedge = .FALSE.

      do 20 i=1,12
         node1id = eledge(elem, i, NODE1)
         node2id = eledge(elem, i, NODE2)
         vect1(X) = nodes(node1id,X) - point(X)
         vect1(Y) = nodes(node1id,Y) - point(Y)
         vect1(Z) = nodes(node1id,Z) - point(Z)
         vect2(X) = nodes(node2id,X) - point(X)
         vect2(Y) = nodes(node2id,Y) - point(Y)
         vect2(Z) = nodes(node2id,Z) - point(Z)
         call normaliz(vect1, uvect1)
         call normaliz(vect2, uvect2)

         if (DABS(dot(uvect1, uvect2) + 1.0) .LT. CAEERR) then
C           The two vectors are on the same line.
            chkedge = .TRUE.
            edge = i
         endif
20    continue

      return
      end

C -----------------------------------------------------------------------
      subroutine bldellst(time, elemf, velf)

C     This procedure builds a list of elements and associated nodes.
C -----------------------------------------------------------------------
      parameter (NODE1=1,NODE2=2,NODE3=3,NODE4=4,NODE5=5,
     1 NODE6=6,NODE7=7,NODE8=8)
      parameter (VELX=1,VELY=2,VELZ=3)

      include "max.f"
      common /ELEMLST/ elemvert(MAXELEMS, 8), elemvel(MAXELEMS, 3)
      integer i, j, numelem, expon, elemID, elemvert
      double precision time, base, vel, elemvel
      character*30 elemf, velf

      open(10, FILE=elemf)
      j = 0

4     format(I6,8(I6))

5     j = j + 1
      read(10,4,END=6) elemID, elemvert(j,NODE1), elemvert(j,NODE2),
```

```
     1 elemvert(j,NODE3), elemvert(j,NODE4), elemvert(j,NODE5),
     2 elemvert(j,NODE6), elemvert(j,NODE7), elemvert(j,NODE8)
       go to 5
6      continue
       numelem = j
       close(10)

       open(11, FILE=velf)

7      format(D15.7)
8      format(3(D15.7))

C      Run through velocity file until right period is found.
9      read(11, 7,END=11) period

       do 10 j=1,numelem
          read(11,8,END=11) elemvel(j,VELX), elemvel(j,VELY),
     1     elemvel(j,VELZ)
10     continue

       if (period .LT. time) then
          go to 9
       else
          continue
       endif

11     continue
       if (period .NE. time) then
          write(*,*) 'Message from BuildElemList: '
          write(*,*) 'WARNING: A match for the time was not found'
       endif

       close(11)
       return
       end

C ---------------------------------------------------------------------
       subroutine rdnewvel(time,velf)

C      This procedure reads in a new block of elemental velocities.
C ---------------------------------------------------------------------
       include "max.f"
       parameter (VELX=1,VELY=2,VELZ=3)
       double precision time
       character*30 velf
       common /ELEMLST/ elemvert(MAXELEMS, 8), elemvel(MAXELEMS, 3)
       common /ELNET/ elside(-1:MAXELEMS, 6, 5),
      1               elused(-1:MAXELEMS,6,4),
      2               eledge(-1:MAXELEMS, 12, 2),
      3               elvel(-1:MAXELEMS, 3)
       integer elside, elused, eledge, elemvert
       double precision elvel, elemvel
       double precision period
       integer j

       open(11, FILE=velf)

7      format(D15.7)
```

```
8     format(3(D15.7))

C     Run through velocity file until right period is found.
      write(*,*) 'Reading new velocities...'
9     read(11, 7,END=11) period

      do 10 j=1,MAXELEMS
         read(11,8,END=11) elemvel(j,VELX), elemvel(j,VELY),
     1    elemvel(j,VELZ)
         if (period .EQ. time) then
            elvel(j,VELX) = elemvel(j,VELX)
            elvel(j,VELY) = elemvel(j,VELY)
            elvel(j,VELZ) = elemvel(j,VELZ)
         endif
10    continue

      if (period .LT. time) then
         go to 9
      else
         continue
      endif

11    continue
      if (period .NE. time) then
         write(*,*) 'Message from BuildElemList: '
         write(*,*) 'WARNING: A match for the time was not found'
      endif

      close(11)
      return
      end

C -------------------------------------------------------------------
      subroutine bldnodls(nodelstf)

C     This procedure builds a nodelist for the element mesh.
C -------------------------------------------------------------------
      parameter (NEXT=0)
      character*30 nodelstf

      include "max.f"
      common /NODELST/ nodelist(MAXNODES, 0:16)
      integer i, j, index, nodelist

3     format(2(I5))
4     format(I7)

      j = 0
      open(10, FILE=nodelstf)
5     j = j + 1
      read(10,3,END=7) index, nodelist(j, NEXT)

      do 6 i=1,nodelist(j,NEXT)-1
         read(10,4) nodelist(j,i)
6     continue
      goto 5

7     continue
```

```
      close(10)
      return
      end

C -----------------------------------------------------------------------
      integer function srchelem (node1,node2,node3,elemID)

C -----------------------------------------------------------------------
      parameter (NEXT=0)
      include "max.f"
      common /NODELST/ nodelist(MAXNODES, 0:16)
      integer node1, node2, node3, elemID, nodelist
      integer i, j, l
      logical found

      srchelem = 0
      found = .FALSE.
      do 80 i=1, (nodelist(node1,NEXT)-1)
10       if (nodelist(node1,i) .NE. elemID) then
20          do 30 j=1,(nodelist(node2,NEXT)-1)
               if (nodelist(node2,j) .EQ. nodelist(node1,i)) then
                  go to 40
               endif
30          continue
40          if (j .LT. nodelist(node2,NEXT)) then
50             do 60 l=1,(nodelist(node3,NEXT)-1)
                  if (nodelist(node3,l) .EQ. nodelist(node1,i)) then
                     go to 70
                  endif
60             continue
70             if (l .LT. nodelist(node3,NEXT)) then
                  srchelem = nodelist(node3,l)
                  found = .TRUE.
                  go to 90
               endif
            endif
         endif
80    continue
90    return
      end

C -----------------------------------------------------------------------
      subroutine bldnet(unit)

C     This procedure loads the network into memeory as elemnet.
C -----------------------------------------------------------------------
      external srchelem

      parameter (ID=5,VELX=1,VELY=2,VELZ=3)
      parameter (NODE1=1,NODE2=2,NODE3=3,NODE4=4)

      include "max.f"
      common /ELNET/ elside(-1:MAXELEMS, 6, 5),
     1               elused(-1:MAXELEMS,6,4),
     2               eledge(-1:MAXELEMS, 12, 2),
     3               elvel(-1:MAXELEMS, 3)
      integer elside, elused, eledge
      double precision elvel
```

```
C     common /NETNODE/ netside(6,5), netused(6,4),
C    1                 netedge(12,2), netvel(3)
      common /ELEMLST/ elemvert(MAXELEMS,8), elemvel(MAXELEMS,3)

      integer netside, netused, netedge, elemvert, srchelem
      double precision netvel, elemvel
      integer j, k, elid

      do 20 j=1,MAXELEMS
         elside(j,1,1) = elemvert(j,1)
         elside(j,1,2) = elemvert(j,5)
         elside(j,1,3) = elemvert(j,6)
         elside(j,1,4) = elemvert(j,2)
         elside(j,2,1) = elemvert(j,2)
         elside(j,2,2) = elemvert(j,6)
         elside(j,2,3) = elemvert(j,7)
         elside(j,2,4) = elemvert(j,3)
         elside(j,3,1) = elemvert(j,3)
         elside(j,3,2) = elemvert(j,7)
         elside(j,3,3) = elemvert(j,8)
         elside(j,3,4) = elemvert(j,4)
         elside(j,4,1) = elemvert(j,4)
         elside(j,4,2) = elemvert(j,8)
         elside(j,4,3) = elemvert(j,5)
         elside(j,4,4) = elemvert(j,1)
         elside(j,5,1) = elemvert(j,5)
         elside(j,5,2) = elemvert(j,8)
         elside(j,5,3) = elemvert(j,7)
         elside(j,5,4) = elemvert(j,6)
         elside(j,6,1) = elemvert(j,2)
         elside(j,6,2) = elemvert(j,3)
         elside(j,6,3) = elemvert(j,4)
         elside(j,6,4) = elemvert(j,1)

         elused(j,1,1) = 1
         elused(j,1,2) = 8
         elused(j,1,3) = 9
         elused(j,1,4) = 5
         elused(j,2,1) = 4
         elused(j,2,2) = 7
         elused(j,2,3) = 12
         elused(j,2,4) = 8
         elused(j,3,1) = 3
         elused(j,3,2) = 6
         elused(j,3,3) = 11
         elused(j,3,4) = 7
         elused(j,4,1) = 2
         elused(j,4,2) = 5
         elused(j,4,3) = 10
         elused(j,4,4) = 6
         elused(j,5,1) = 3
         elused(j,5,2) = 4
         elused(j,5,3) = 1
         elused(j,5,4) = 2
         elused(j,6,1) = 11
         elused(j,6,2) = 10
         elused(j,6,3) = 9
         elused(j,6,4) = 12
```

```
            eledge(j,1,1) = elemvert(j,5)
            eledge(j,1,2) = elemvert(j,6)
            eledge(j,2,1) = elemvert(j,8)
            eledge(j,2,2) = elemvert(j,5)
            eledge(j,3,1) = elemvert(j,7)
            eledge(j,3,2) = elemvert(j,8)
            eledge(j,4,1) = elemvert(j,6)
            eledge(j,4,2) = elemvert(j,7)
            eledge(j,5,1) = elemvert(j,5)
            eledge(j,5,2) = elemvert(j,1)
            eledge(j,6,1) = elemvert(j,8)
            eledge(j,6,2) = elemvert(j,4)
            eledge(j,7,1) = elemvert(j,7)
            eledge(j,7,2) = elemvert(j,3)
            eledge(j,8,1) = elemvert(j,6)
            eledge(j,8,2) = elemvert(j,2)
            eledge(j,9,1) = elemvert(j,1)
            eledge(j,9,2) = elemvert(j,2)
            eledge(j,10,1) = elemvert(j,4)
            eledge(j,10,2) = elemvert(j,1)
            eledge(j,11,1) = elemvert(j,3)
            eledge(j,11,2) = elemvert(j,4)
            eledge(j,12,1) = elemvert(j,2)
            eledge(j,12,2) = elemvert(j,3)

            elvel(j,VELX) = elemvel(j,VELX)
            elvel(j,VELY) = elemvel(j,VELY)
            elvel(j,VELZ) = elemvel(j,VELZ)

            do 10 k=1,6
               elside(j,k,ID) = srchelem(elside(j,k,NODE1),
     1           elside(j,k,NODE2),elside(j,k,NODE3), j)
10          continue

20       continue

         return
         end

C ------------------------------------------------------------------------
         subroutine runelem(time)

C ------------------------------------------------------------------------
         double precision time

         common /FILES/ elemf, velf, elemnetf, nodelstf,
        1               nodef, sumf, outf, inf
         character*30 elemf, velf, elemnetf, nodelstf,
        1             nodef, sumf, outf, inf
         external bldellst, bldnodls, bldnet

         call bldellst(time, elemf, velf)
         call bldnodls(nodelstf)

         open(10,FILE=elemnetf)
         call bldnet(10)
         close(10)
```

```
      return
      end
C -------------------------------------------------------------------
```

```
C -----------------------------------------------------------------
C Program          : 3D Particle Tracking System - FORTRAN77 version
C Program Segment  : PTSIM.F
C Description      : This program simulates the travel of a radio-
C                    active particle from an initial element to a
C                    boundary of the test region. It handles the case
C                    where a particle attempts to move from one
C                    element into another that has an opposing travel
C                    vector and thus moves along the boundary between
C                    two neighbouring elements to an edge or a node.
C                    It is assumed that the data has been set up in
C                    the required manner for reading into the records.
C                    Velocities are assumed to be defined in a step-
C                    wise fashion and a new set is read from the disk
C                    file every time the stated period has been reached.
C                    This is a translated version of the program. The
C                    original was written in Turbo Pascal to run on
C                    an IBM PC. The second version was written in
C                    IRIS-4D Pascal to run on the Silicon Graphics 4D
C                    super-mini computer. This version is in standard
C                    FORTRAN 77.
C Last Revision    : February 6, 1991
C Revised by       : Vanessa Williams
C                    SOMAPRO Ltd.
C Authors          : Arthur Chan, Chris Black, Peter Ng,
C                    Vanessa Williams
C
C Copyright (c) 1991 Ontario Hydro
C -----------------------------------------------------------------
C -----------------------------------------------------------------
      program ptsim

C     The main line of this program begins by finding a starting point
C     and then moving through that element to its side. If the velocity
C     vector in the next element allows motion into that element, the
C     simulation then moves through the next element, and so on. If, at
C     any time, the next velocity vector does not allow motion into the
C     next element, the program calculates the resultant velocity in
C     the plane of the side and travels on that plane to an edge. Once
C     at an edge, each of the two elements that would represent forward
C     motion are tested for favourable velocity vectors. If a favour-
C     able velocity is found the program moves through that element.
C     If not, the resultant along the edge line is found and the
C     program travels along the edge to a node. At the node, each of
C     the four elements that would represent forward motion are tested
C     for favourable velocity vectors and are moved into if appro-
C     priate. If not, each available side is tested and moved into if
C     appropriate. If again, not feasible, each of the availble edges
C     are tested and moved along if appropriate. If still no suitable
C     direction has been found for motion, the particle is assumed
C     to be blocked and the simulation is ended. Any time when the
C     program is at a node, the surrounding elements are checked to
C     see if that node is at the boundary, indicating the escape of
C     the studied particle.
C        After each successful move, the exit point and element
C     number and accumulated travel time are recorded, along with
C     the value of the atnode and atedge flags.
C -----------------------------------------------------------------
```

```
      include "header.f"

      lasttime = 0
      elvel(-1, X) = 0.0
      elvel(-1, Y) = 0.0
      elvel(-1, Z) = 0.0
      elvel(0, X) = 0.0
      elvel(0, Y) = 0.0
      elvel(0, Z) = 0.0

      call rdconfig()
      call setupsum(sumf)
      write(*,*) 'Building element vault (this will take a minute)...'
      call rdnodeco(nodes, nodef)
      call rdnolst(nolstarr, nodelstf)
      call runelem(lasttime)
      write(*,*) 'FINISHED.'
      open(8,FILE=outf)
      open(9,FILE=inf)

10    lasttime = 0
      call readinit(more,elemid,startpt,vperiod,9)

      if (more .EQ. .TRUE.) then
         call prbanner(8)
         tottime = 0.0
         totdist = 0.0
         lastelem = 0
         atnode = .FALSE.
         atedge = .FALSE.
         escape = .FALSE.
         blocked = .FALSE.
         firstel = .TRUE.

         if (chkedge(startpt,elemid,edge) .EQ. .TRUE.) then
            atedge = .TRUE.
         endif
         if (chknode(startpt,elemid,presnode) .EQ. .TRUE.) then
            atnode = .TRUE.
         endif

C 20       format(3(F13.2),I8,2(L7),2(D9.2))
20       format(3(F10.2),I8,2(L7),2(D13.4))
21       format(I6,3(F9.2))
22       format(A16,I6,A15,D8.2,2(A3,D8.2),A3)

C        Write initial output record to detail and summary files
         write(8,20) startpt(X), startpt(Y), startpt(Z), elemid,
     1               atedge, atnode, tottime, totdist
C         write(7,21) elemid, startpt(X), startpt(Y), startpt(Z)
         sumid = elemid
         sumstart(X) = startpt(X)
         sumstart(Y) = startpt(Y)
         sumstart(Z) = startpt(Z)
         write(*,*) 'Tracking element', elemid, ' from coord: x=',
     1      startpt(X), ' y=', startpt(Y), ' z=', startpt(Z), '...'

30       if (.NOT.((escape .EQ. .TRUE.) .OR.
```

```
     1            (blocked .EQ. .TRUE.))) then
C          ------------------------------------------------------
C          If the particle is in an element or on the side, try
C          moving through the element to another side
C          ------------------------------------------------------
           if ((.NOT. atedge) .AND. (.NOT. atnode) .AND.
     1          thruelem(startpt,elemid,endpt,endside,nextelem,
     2                   travtime,lastelem)) then
              tottime = tottime + travtime
              do 40 i=1,3
                 oldvel(i) = elvel(elemid,i)
                 exitpt(i) = endpt(i)
                 vector(i) = exitpt(i) - startpt(i)
                 startpt(i) = exitpt(i)
40            continue
              totdist = totdist + magnitud(vector)
              exitside = endside
              startside = exitside
              atedge = .FALSE.
              atnode = .FALSE.
              lastelem = elemid
              elemid = nextelem
C          ------------------------------------------------------
C          If the particle is on the side of an element and it
C          cannot move through the element, try moving through the
C          adjacent element
C          ------------------------------------------------------
           else if ((.NOT. atedge) .AND. (.NOT. atnode) .AND.
     1              (travtime .EQ. 0) .AND.
     2              thruelem(startpt,nextelem,exitpt,endside,
     3                       nxtelem,travtime,lastelem)) then
              tottime = tottime + travtime
              do 50 i=1,3
                 oldvel(i) = elvel(elemid,i)
                 vector(i) = exitpt(o) - startpt(i)
                 startpt(i) = exitpt(i)
50            continue
              totdist = totdist + magnitud(vector)
              exitside = endside
              startside = exitside
              atedge = .FALSE.
              atnode = .FALSE.
              lastelem = nextelem
              elemid = nxtelem
C          ------------------------------------------------------
C          If the particle cannot move through the element to another
C          side, try moving along the side of the element to an edge
C          ------------------------------------------------------
           else
              if (firstel) then
                 startside = endside
                 lastelem = elemid
                 firstel = .FALSE.
              endif
           if ((.NOT. atedge) .AND. (.NOT. atnode)) then
              elemid = lastelem
              if (alongsid(startpt,elemid,startside,endpt,vel,
     1                     exitedge,travtime)) then
```

```
C                 ------------------------------------------------
C                 On a side but not an edge; can't go into element
C                 but can move along the side - go ahead and move
C                 ------------------------------------------------
                  tottime = tottime + travtime
                  do 60 i=1,3
                     oldvel(i) = vel(i)
                     exitpt(i) = endpt(i)
                     vector(i) = exitpt(i) - startpt(i)
                     startpt(i) = exitpt(i)
60                continue
                  totdist = totdist + magnitud(vector)
                  edge = exitedge
                  atedge = .TRUE.
                  atnode = .FALSE.
               else
C                 ------------------------------------------------
C                 On a side, but unable to move along side
C                 ------------------------------------------------
                  blocked = .TRUE.
               endif
            else if (atedge .AND. (.NOT. atnode)) then
               curlarge = 0.0
               travtime = 0
C              ------------------------------------------------------
C              The particle is at an edge, now try moving through an
C              element to a side
C              ------------------------------------------------------
               if (getelem(eledge(lastelem,edge,NODE1),
     1            eledge(lastelem,edge,NODE2),startpt,oldvel,t,endpt,
     2            endside,bestelem,bestnext,largemag,lastelem)) then
C                 ---------------------------------------------------
C                 The element we have chosen is valid for motion
C                 ---------------------------------------------------
                  curlarge = largemag
                  travtime = t
                  elemid = bestelem
                  nextelem = bestnext
                  do 70 i=1,3
                     exitpt(i) = endpt(i)
                     bestvel(i) = vel(i)
70                continue
                  exitside = endside
                  bsnxedge = 0
                  atedge = .FALSE.
                  atnode = .FALSE.
               endif

C              ------------------------------------------------------
C              See if moving along the side to another edge is a
C              better move
C              ------------------------------------------------------
               if (getside(lastelem,edge,oldvel,startpt,t,mainelem,
     1                     endedge,endpt,vel,largemag,endside) .AND.
     2             (largemag .GT. curlarge)) then
C                 ---------------------------------------------------
C                 The side chosen is the best choice so far
C                 ---------------------------------------------------
```

```
                  curlarge = largemag
                  travtime = t
                  elemid = mainelem
                  nextelem = -1
                  bsnxedge = endedge
                  exitside = endside
                  do 80 i=1,3
                     exitpt(i) = endpt(i)
                     bestvel(i) = vel(i)
 80               continue
                  atedge = .TRUE.
                  atnode = .FALSE.
               endif

C              ------------------------------------------------------
C              The particle is on an edge, see if moving along the
C              edge to a node is a better move
C              ------------------------------------------------------
               call alongedg(edge,lastelem,startpt,endpt,vel,t)
               if ((dot(oldvel,vel) .GE. 0) .AND.
     1              (magnitud(vel) .GT. curlarge) .AND. (t .GT. 0)) then
C                 ---------------------------------------------------
C                 Move along edge is best option
C                 ---------------------------------------------------
                  curlarge = magnitud(vel)
                  nextelem = -1
                  travtime = t
                  elemid = lastelem
                  do 90 i=1,3
                     exitpt(i) = endpt(i)
                     bestvel(i) = vel(i)
 90               continue
                  bsnxedge = 0
                  exitside = 0
                  atedge = .FALSE.
                  atnode = .TRUE.
                  if ((DABS(nodes(eledge(elemid,edge,NODE2),X)
     1                    -exitpt(X)) .LT. POINTERR) .AND.
     2                (DABS(nodes(eledge(elemid,edge,NODE2),Y)
     3                    -exitpt(Y)) .LT. POINTERR) .AND.
     4                (DABS(nodes(eledge(elemid,edge,NODE2),Z)
     5                    -exitpt(Z)) .LT. POINTERR)) then
                     presnode = eledge(elemid,edge,NODE2)
                  else
                     presnode = eledge(elemid,edge,NODE1)
                  endif
               endif

C              ------------------------------------------------------
C              Update parameters if a successful move was made from
C              the edge
C              ------------------------------------------------------
               if (curlarge .GT. 0) then
                  tottime = tottime + travtime
                  exitedge = bsnxedge
                  edge = exitedge
                  lastelem = elemid
                  do 100 i=1,3
```

```
                  oldvel(i) = bestvel(i)
                  vector(i) = exitpt(i) - startpt(i)
                  startpt(i) = exitpt(i)
100               continue
                  totdist = totdist + magnitud(vector)
                  if (nextelem .GT. 0) then
                     elemid = nextelem
                  endif
                  startside = exitside
               else
                  blocked = .TRUE.
               endif

            else if (atnode) then
               curlarge = 0
C              ------------------------------------------------------
C              Particle is now at a node, see if the particle can
C              move through a surrounding element to a side
C              ------------------------------------------------------
               if (elemfr(presnode,startpt,oldvel,elem,nextel,t,
     1                    endpt,endside,largemag,lastelem) .AND.
     2             (largemag .GT. curlarge)) then
                  curlarge = largemag
                  elemid = elem
                  nextelem = nextel
                  travtime = t
                  do 110 i=1,3
                     bestvel(i) = elvel(elemid,i)
                     exitpt(i) = endpt(i)
110               continue
                  exitside = endside
                  atedge = .FALSE.
                  atnode = .FALSE.
               endif

C              ------------------------------------------------------
C              See if travelling from the node along the side of the
C              element to an edge is a better move
C              ------------------------------------------------------
               if (sidefr(presnode,oldvel,t,mainelem,endedge,endpt,
     1                    vel,largemag,endside) .AND.
     2             (largemag .GT. curlarge)) then
C                 ---------------------------------------------------
C                 Have successfully moved along a side and it was
C                 better than the attempt made through the element
C                 ---------------------------------------------------
                  travtime = t
                  elemid = mainelem
                  nextelem = -1
                  curlarge = largemag
                  do 120 i=1,3
                     bestvel(i) = vel(i)
                     exitpt(i) = endpt(i)
120               continue
                  edge = endedge
                  exitedge = edge
                  atedge = .TRUE.
                  atnode = .FALSE.
```

```
                endif

C               ------------------------------------------------------
C               See if travelling from the node along an adjacent edge
C               to another node provides a better move
C               ------------------------------------------------------
                if (getedge(presnode,oldvel,mainelem,t,endpt,vel,
     1                      exitnode,largemag,endedge) .AND.
     2              (largemag .GT. curlarge)) then
                   travtime = t
                   elemid = mainelem
                   nextelem = -1
                   curlarge = largemag
                   do 130 i=1,3
                      bestvel(i) = vel(i)
                      exitpt(i) = endpt(i)
130                continue
                   presnode = exitnode
                   edge = endedge
                   exitedge = 0
                   atedge = .FALSE.
                   atnode = .TRUE.
                endif

C               ------------------------------------------------------
C               Update parameters if a suitable move from the node has
C               been made
C               ------------------------------------------------------
                if (curlarge .GT. 0) then
                   tottime = tottime + travtime
                   do 140 i=1,3
                      oldvel(i) = bestvel(i)
                      vector(i) = exitpt(i) - startpt(i)
                      startpt(i) = exitpt(i)
140                continue
                   totdist = totdist + magnitud(vector)
                   lastelem = elemid
                   if (nextelem .GT. 0) then
                      elemid = nextelem
                   endif
                   startside = exitside
                else
                   blocked = .TRUE.
                endif
             endif
             endif

C            ------------------------------------------------------------
C            Check to see if the particle has escaped
C            ------------------------------------------------------------
             if (atnode) then
                escape = nodebnd(presnode,lastelem)
             else if (atedge) then
                do 150 i=0,16
                   nodearr1(i) = nolstarr(eledge(lastelem,edge,NODE1),i)
                   nodearr2(i) = nolstarr(eledge(lastelem,edge,NODE2),i)
150             continue
                escape = edgebnd(edge,nodearr1,nodearr2,lastelem)
```

```
            else
               if (elside(lastelem,startside,ID) .LT. 1) then
                  escape = .TRUE.
               else
                  escape = .FALSE.
               endif
            endif

C           ------------------------------------------------------------
C           If the particle has escaped or is blocked, set the
C           correct flags and output messages accordingly
C           ------------------------------------------------------------
C 170         format(3(F9.2),F8.1,A9)
170         format(I6,6(F9.2),F8.1,A9)
            if (escape .EQ. .TRUE.) then
               blocked = .FALSE.
C               write(7,170) startpt(X),startpt(Y),startpt(Z),
C     1                      tottime/31536000, '  escaped'
               write(7,170) sumid, sumstart(X), sumstart(Y),
     1            sumstart(Z), startpt(X),startpt(Y),startpt(Z),
     2            tottime/31536000, '  escaped'
               write(8,20) startpt(X), startpt(Y), startpt(Z), elemid,
     1               atedge, atnode, tottime, totdist
               write(8,*) 'Particle escaped...simulation ended'
            endif

            if (blocked) then
C               write(7,170) startpt(X),startpt(Y),startpt(Z),
C     1                      tottime/31536000, '  blocked'
               write(7,170) sumid, sumstart(X), sumstart(Y),
     1            sumstart(Z), startpt(X),startpt(Y),startpt(Z),
     2            tottime/31536000, '  blocked'
               write(8,*) 'Particle blocked...simulation ended'
            endif

C           ------------------------------------------------------------
C           If the particle is not blocked or has not escaped, check
C           to see if we have moved to an edge or node and set any
C           flags as required
C           ------------------------------------------------------------
            if ((.NOT. (escape .EQ. .TRUE.)) .AND.
     1          (.NOT. (blocked .EQ. .TRUE.))) then
               if ((.NOT.(atedge .OR. atnode)) .AND.
     1             chkedge(exitpt,lastelem,exitedge)) then
C                  ---------------------------------------------------
C                  By some fluke we have arrived exactly at an edge
C                  ---------------------------------------------------
                  atedge = .TRUE.
                  elemid = lastelem
                  edge = exitedge
               endif

               if ((.NOT. atnode) .AND.
     1             chknode(exitpt,lastelem,exitnode)) then
C                  ---------------------------------------------------
C                  By some fluke we have arrived exactly at an node
C                  ---------------------------------------------------
                  atnode = .TRUE.
```

```
                  presnode = exitnode
                  elemid = lastelem
               endif

C              ------------------------------------------------------
C              Write output record
C              ------------------------------------------------------
               write(8,20) startpt(X), startpt(Y), startpt(Z), elemid,
     1               atedge, atnode, tottime, totdist

C              ------------------------------------------------------
C              Read new velocities if required
C              ------------------------------------------------------
               if (tottime .GE. (lasttime + vperiod)) then
                  lasttime = lasttime + vperiod
                  call rdnewvel(lasttime,velf)
               endif
            endif

            firstel = .FALSE.
            go to 30
         endif
      endif

C     REPEAT UNTIL NOT MORE
      if (more) then
         go to 10
      endif

      close(7)
      close(8)

      stop
      end
```

```
      integer MAXELEMS, MAXNODES
C     parameter (MAXELEMS=14704, MAXNODES=16458)
      parameter (MAXELEMS=15586, MAXNODES=17378)
```

APPENDIX III

System Manual

3D Particle Tracking System

1. Introduction

The intention of the System Manual, for the 3D Particle Tracking System, is to provide a system overview and to act as a programmers reference. Information provided in this document should facilitate system maintenance and simplify future enhancements. The scope of this document includes: system requirements, algorithm outline, and system development.

2. System Requirements

The system was originally written in the Pascal implementation released by Silicon Graphics for the IRIS 4D/2405 super-mini computer system. All source code was written using standard ANSI Pascal, with the exception of one program, "tdelmlst.p", which utilizes Turbo Pascal (for its string handling capabilities). A more recent version of the system has been developed entirely in FORTRAN 77. Although the Silicon Graphics F77 compiler provides some extensions over ANSI FORTRAN 77, care has been taken to avoid using these features. Both versions of the system are currently running in a UNIX environment (Turbo Pascal in DOS only).

The memory and disk storage requirements vary with the size of the mesh, but a mesh of 15000 elements can be operated with nine (9) megabytes of RAM and four (4) megabytes of disk space.

A version of the system also exists in Borland's Turbo Pascal implementation for micro-computers operating in a DOS environment. This version is extremely limited, reducing the number of possible elements in a given mesh to several hundred. If necessary, the code could be modified to operate with larger meshes on a micro by reading the information for each element from disk as it is required, rather than storing the information for all elements in RAM. This however would result in a significant decrease in performance time.

An older version of the system exists for use with two-dimensional meshes and it is available only in the Turbo Pascal version.

The remainder of this document describes the FORTRAN 77 version of the system only.

3. Algorithm Overview

An outline of the Ontario Hydro algorithm is presented here. This can be used as a reference for the corresponding program structures and subroutines that follow. A flow chart of the algorithm is provided in Appendix III.a.

The algorithm is as follows:

- Move through the present element to another side;
- If the particle cannot move through the current element, try the adjacent element;
- If possible, move into and through the next element and repeat;

- If not possible, move along the side between adjacent elements until an edge is reached;
- As we are now at an edge, try to move through one of the elements around the edge;
- Also try moving onto and across one of the sides adjoining the present edge: if this move has a larger velocity vector and velocity dot product with the old velocity, take this route rather than the previous element move;
- Also try moving along the edge itself to a node: if this is better than the previous two moves, do this instead;
- If we have moved along an edge to a node, try moving through an element surrounding the node;
- Also try moving onto and across one of the sides adjoining the node: if this move is better, do this instead;
- Also try moving along each of the edges to another node: if this is better than the previous two routes, do this instead.
- After each move, check to see that we have not inadvertently arrived at a node or edge. If so, proceed with the steps for being at a node or edge respectively.
- If, at any time, we are unable to make a further move, report the particle as being blocked and end the simulation.

4. System Development

4.1 Program Structure

The system is composed of two executable program modules:

_ ptsim.exe	(particle tracking simulation)
_ ptutil.exe	(particle tracking utilities)

Ptutil is responsible for translating the AECL test data into a format usable by the primary program module, ptsim. See the *User Manual* for further information.

Ptsim is composed of three program segments: header.f, function.f, and ptsim.f.

1. **Header.f** contains common storage declarations and data declarations for the main program unit. It is included in the source for ptsim at compile time.
2. **Function.f** contains all functions and procedures.
3. **Ptsim.f** is the main program containing the algorithm for pathway analysis.

An additional file, max.f, contains constant declarations specifying the size of the mesh. This file may need to be edited each time a new element mesh is studied. A makefile has been created to simplify compiling and linking the various program segments. Program compilation is discussed in the *User Manual*.

4.2 File Dependencies and Inputs/Outputs

Data files for constructing the rock host mass, defining the element within the host, and associated velocity gradients are obtained from AECL. The AECL files must be converted into a secondary format before the system can be used.

The following conventions are used as file descriptors:

_<filename> If the file is an AECL input file, the filename will be determined by AECL. If the file is a system input file, the filename is determined by the user.
_<x> This descriptor indicates that a numerical value should be attached to the filename associated with it. The number chosen is determined by the user - generally is ascending order depending on the number of test problems to execute.

4.2.1 AECL Data Files

The input files required from AECL are:
_ <filename>.in
_ <filename>.geo
_ <filename>.res

<filename>.in File containing initial locations of each of the particles within the given test problem. The file also contains the names of the corresponding .geo and .res files, the number of paths in the problem, and the names of the path files corresponding to each initial location.

<filename>.geo Geometry file in MOTIF format for a test problem. This file includes global nodal coordinates and mesh connectivities ie. nodes which make up the mesh and their respective coordinates (x,y,z); and elements along with their associated nodes. The .geo file is used to generate three files: nodes<x>.dat, elements<x>.dat, and nodelist<x>.dat for input to the system. In addition, the number of nodes and number of elements for the mesh is found in this file.

<filename>.res Velocity distribution file in MOTIF format for a given test problem. This file contains nodal head values, nodal temperature values and elemental velocities. We are only concerned with the elemental velocities. This file is used to generate the file velocity<x>.dat.

See Appendix III.b for sample .in, .geo, .res files and their formats.

4.2.2 3D Particle Tracking System Data Files

The input files the 3D Particle Tracking system requires are:
_ velocity<x>.dat
_ nodes<x>.dat
_ elements<x>.dat
_ nodelist<x>.dat
_ <filename>.cfg
_ <filename>.init

velocity<x>.dat Velocity file containing the velocity vectors for each element of a given mesh. This file must be generated by the user from the <filename>.res file.

nodes<x>.dat Node file containing node ID's and their respective coordinates. This file is generated by ptutil from the <filename>.geo file.

elements<x>.dat Element file containing element ID's and associated nodes for each element. This file is generated from <filename>.geo by ptutil.

nodelist<x>.dat This file contains a list of node ID's and their associated elements. This file is also created by the ptutil program from <filename>.geo.

<filename>.cfg This file must be created by the user. It is a configuration file which contains a list of all input/output files that the system requires.

<filename>.init This file contains a list of the starting points for pathway analysis. It must be created by the user from AECL's .in file. The format is as follows: element id, x-coord, y-coord, z-coord.

These files are assigned their actual names (which the user can change at will - these names are not sacred) in the <filename>.cfg file. The name of the configuration file itself is supplied to the program by the user at run time.

Appendix III.b provides samples of all of these files and their exact formats.

The output files generated from the 3D Particle Tracking system are:

- _ summary<x>
- _ detail<x>

summary<x> This file contains a summary of all paths traversed including their initial start point, exit point, time travelled, and exit status (escaped or blocked).

detail<x> This file is a compilation of the specific paths traversed for all test cases for a test problem.

Sample output files are provided in Appendix III.b.

4.3 System Utilities

The ptutil program takes AECL's geometry file and produces the files nodes<x>.dat, elements<x>.dat, and nodelist<x>.dat (inputs to ptsim). Inputs to the utility program are the pathname of the .geo file and the pathname of the user-created .cfg file. (Note that the same .cfg file is used by both the main program and the utility program).

4.4 Transient Velocity Problems

The particle tracking system was originally implemented assuming a steady-state problem in which elemental velocities remained constant throughout the simulation. The program can handle problems involving gradients which change over time if the following conditions are met:

1. The velocity.dat file contains a block of velocities for each time period; each block preceded by the corresponding start time, i.e. the time at which the new values will take effect.

2. The **readinit** procedure is modified to set the variable **vperiod** to the appropriate time interval. This interval specifies when the elemental velocities will change. Currently, **vperiod** is set to an arbitrarily large number to simulate a problem in which the velocities never change.

The algorithm checks the elapsed time after each successful move of the particle being tracked (through an element, along a side or an edge). If the elapsed time since the last set of velocities was read is greater than **vperiod**, the next block of gradients is read from the disk. Note that since the program is designed to make only these discrete moves, it is not possible to change the velocity distribution vectors while the particle is part way through its move. The error introduced by this method should be minimal.

4.5 Procedure Descriptions

Following is an overview of the functions and procedures active in the particle tracking system. The routines are classified as: Main, support, utility, and input/output.

4.5.1 Input/Output

The input/output routines are generally responsible for all file I/O and screen I/O. This includes construction and storage in RAM of various data structures required by the system.

rdnodeco

type:	subroutine;
inputs:	nodefile name;
outputs:	array of node ID's and their respective coordinates
description:	reads the file that contains the coordinates for each of the nodes. It assumes that the records of the file are ordered by the node number so that later reference to a specific node can use the array index.
invokes:	none;
invoked by:	main.

rdnolst

type:	subroutine;
inputs:	nodelistfile name;
outputs:	nodelist;
description:	builds a list that stores the node ID's and element ID's that have that node in common.
invokes:	none;
invoked by:	main.

runelem

type:	subroutine;
inputs:	none;
outputs:	time;
description:	creates the element and node lists. The time returned is the time for successive velocity sets; the current system assumes a steady-state problem.
invokes:	bldellst(), bldnolst(), bldnet();
invoked by:	main.

readinit

type:	subroutine;
inputs:	input file unit number;
outputs:	more, elemid, position, velocity period;
description:	reads the initial start coordinates for pathway analysis from the user generated input file <filename>.in. The current program assumes a steady-state problem, thus velocity period is set to a large value. If a transient problem is desired to be tested, this routine should be changed to accommodate this.
invokes:	none;
invoked by:	main.

rdnewvel

type:	subroutine;
inputs:	time, velocity file name;
outputs:	none;
description:	reads new block of elemental velocities;
invokes:	none;
invoked by:	main.

4.5.2 Utility

Utility routines facilitate common tasks required in the various functions and procedures.

normaliz

type:	subroutine;
inputs:	vector;
outputs:	unitvector;
description:	an input vector is normalised and the resulting unit vector is returned.
invokes:	magnitud();
invoked by:	multiple.

dot

type:	function double;
inputs:	vector1, vector2;
outputs:	dot product;
description:	the dot product of two input vectors is returned;
invokes:	none;
invoked by:	multiple.

max

type:	function double;
inputs:	value1, value2;
outputs:	largest value;
description:	the largest of value1 and value2 is returned;
invokes:	none;
invoked by:	multiple.

min

type:	function double;
inputs:	value1, value2;
outputs:	smallest value;
description:	the smallest of value1 and value2 is returned;
invokes:	none;
invoked by:	multiple.

magnitud

type:	function double;
inputs:	vector;
outputs:	magnitude of vector;
description:	the magnitude of the input vector is returned;
invokes:	none;

invoked by: multiple.

4.5.3 Support

The following routines provide a support function to the functions and procedures used to implement the algorithm.

nodebnd
type: function logical;
inputs: currnode, elemid;
outputs: logical;
description: checks if a given node is on the boundary of the defined set of elements;
invokes: none;
invoked by: main.

edgebnd
type: function logical;
inputs: curredge, elemid, nodelst1, nodelst2;
outputs: logical;
description: checks if a given edge lies on the boundary edge of the defined set of elements;
invokes: none;
invoked by: main.

chkedge
type: function logical;
inputs: point, element;
outputs: edge;
description: checks if the point given is on one of the edges of the specified element;
invokes: normaliz();
invoked by: main.

chknode
type: function logical;
inputs: point, element;
outputs: nodeid;
description: checks if the point given is on one of the nodes of the specified element;
invokes: none;
invoked by: main.

bldellst
type : subroutine;
inputs: time, element and velocity file names;
outputs: none;
description: reads the elements<x>.dat file and the velocity<x>.dat file and builds the element list;
invokes: none;
invoked by: runelem().

bldnodls

type: subroutine;
inputs: node list filename;
outputs: none;
description: reads the nodelist<x>.dat file and builds the node lists;
invokes: none;
invoked by: runelem().

bldnet
type: subroutine;
inputs: none;
outputs: none;
description: builds the element network vault from the element and node lists;
invokes: srchelem();
invoked by: runelem().

srchelem
type: function integer;
inputs: node1, node2, node3, elemid;
outputs: none;
description: searches the node list for the given element id;
invokes: none;
invoked by: bldnet().

ixedge
type: function logical;
inputs: startpt, velocity, node1, node2;
outputs: endpt, time;
description: tests the edge defined by the given nodes to see if the particle, in the direction of the velocity vector, will intersect that edge;
invokes: min(), max();
invoked by: alongsid().

ixside
type: function logical;
inputs: startpt, velocity, node1, node2, node3;
outputt: endpt, time;
description: tests the side defined by the given nodes to see if the particle, in the direction of the velocity vector, will intersect that side;
invokes: none;
invoked by: thruelem().

corange
type: subroutine;
inputs: element;
outputs: max, min;
description: determines the largest and smallest x, y, z coordinates used for the nodes of the given element;
invokes: none;
invoked by: thruelem().

findvel
type: subroutine;
inputs: pt1, pt2, pt3, vel1, vel2;
outputs: planevel;

description: finds the direction and magnitude of the resultant velocity vector in the plane between two elements;
invokes: normaliz(), dot(), max();
invoked by: alongsid().

resvel
type: subroutine;
inputs: node1, node2;
outputs: velmag;
description: finds the resultant velocity along an edge by summing the projected velocity components of the surrounding elements;
invokes: normaliz(), dot();
invoked by: alongedg().

4.5.4 Main

The primary functions and procedures used to implement the algorithm follow.

thruelem

type:	function logical;
inputs:	startpt, elemid;
outputs:	exitpt, exitside, nextelem, prevelem, travtime;
description:	the best move through an element to another side is returned if possible otherwise the function returns false;
invokes:	ixside(), corange();
invoked by:	main, getelem(), elemfr().

alongsid

type:	function logical;
inputs:	startpt, elemid, side;
outputs:	exitpt, velocity, exitedge, travtime;
description:	the best move along the side of an element to an edge is returned if possible otherwise the function returns false;
invokes:	ixedge(), findvel();
invoked by:	main, getside(), sidefr().

alongedg

type:	subroutine;
inputs:	elemid, curredge, startpt;
outputs:	exitpt, velocity, travtime;
description:	the best move along the edge of an element to a node is returned;
invokes:	resvel(), normaliz();
invoked by:	main, getedge().

getelem

type:	function logical;
inputs:	node1id, node2id, startpt, oldvel, prevelem;
outputs:	elemid, nextelem, exitpt, exitside, travtime, largemag;
description:	finds which element, surrounding the edge we are on, that provides the best path for further motion through that element to a side;
invokes:	thruelem(), dot();
invoked by:	main.

getside

type:	function logical;
inputs:	elemid, edge, startpt, lastvel;
outputs:	chiefel, endedge, exitpt, exitside, velocity, travtime, largemag;
description:	finds which side, surrounding the edge we are on, that provides the best path for further motion along that elements side to another edge;
invokes:	alongsid(), dot();
invoked by:	main.

elemfr

type:	function logical;
inputs:	node, startpt, oldvel, prevelem;
outputt:	elemid, nextelem, exitpt, exitside, travtime, largemag;

description: finds which element, surrounding the node we are on, that provides the best path for further motion through that element to a side;
invokes: thruelem(), dot();
invoked by: main.

sidefr
type: function logical;
inputs: node, last velocity;
outputs: chiefel, exitpt, exitedge, exitside, velocity, travtime, largemag;
description: finds which side, surrounding the node we are on, that provides the best path for further motion along that elements side to an edge;
invokes: alongsid(), dot();
invoked by: main.

getedge
type: function logical;
inputs: node, last velocity;
outputs: chiefel, endpt, exitnode, endedge, velocity, travtime, largemag;
description: finds which edge, surrounding the node we are on, that provides the best path to move along to another node;
invokes: alongedg(), dot();
invoked by: main.

4.5.5 Miscellaneous

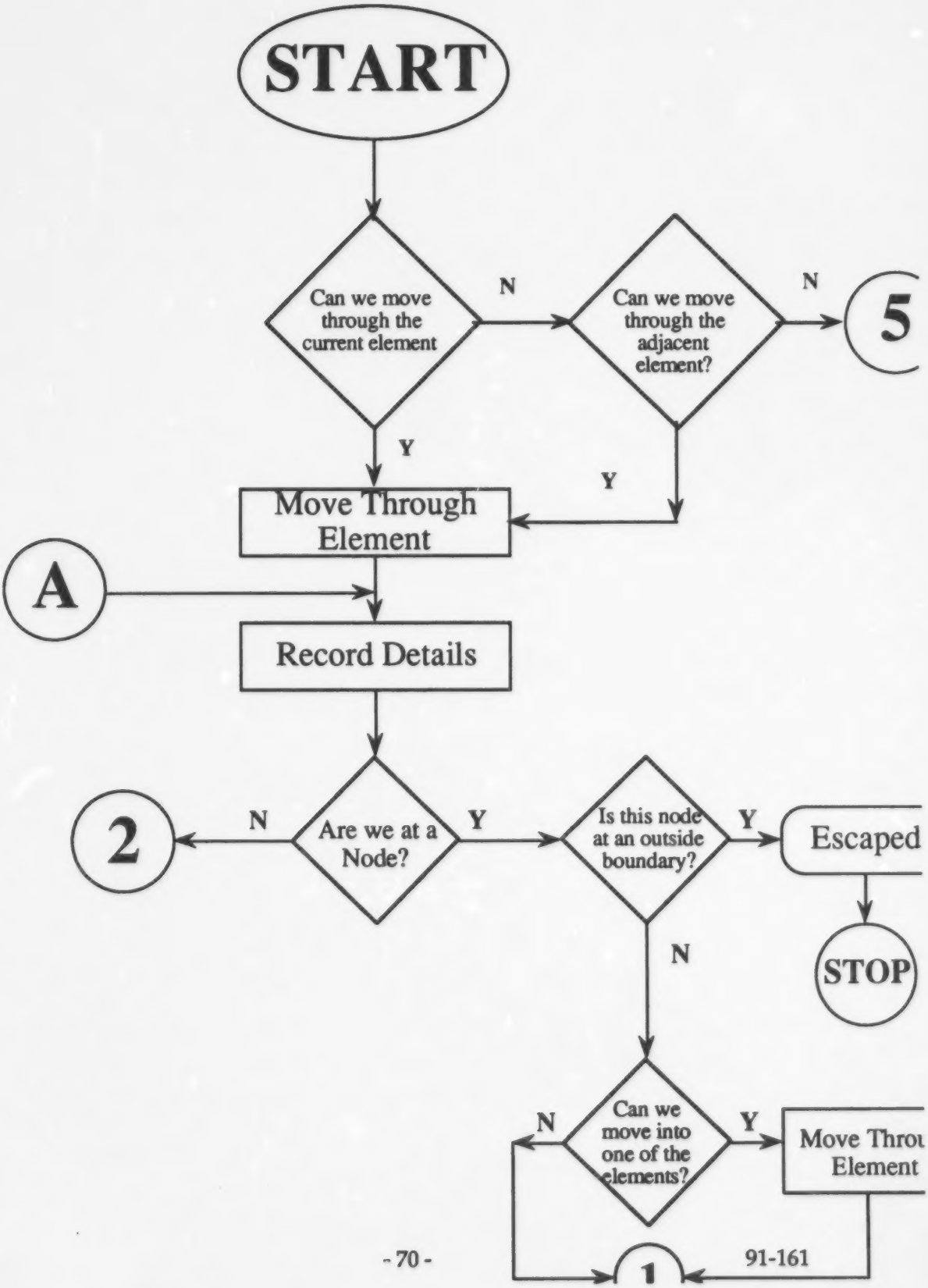
prbanner
type: subroutine;
inputs: none;
outputs: none;
description: prints the heading banner for each of the pathway tests in the detail file;
invokes: none;
invoked by: main.

setupsum
type: subroutine;
inputs: sumfile: name of summary file;
outputs: none;
description: sets up summary file and prints associated banner;
invokes: none;
invoked by: main.

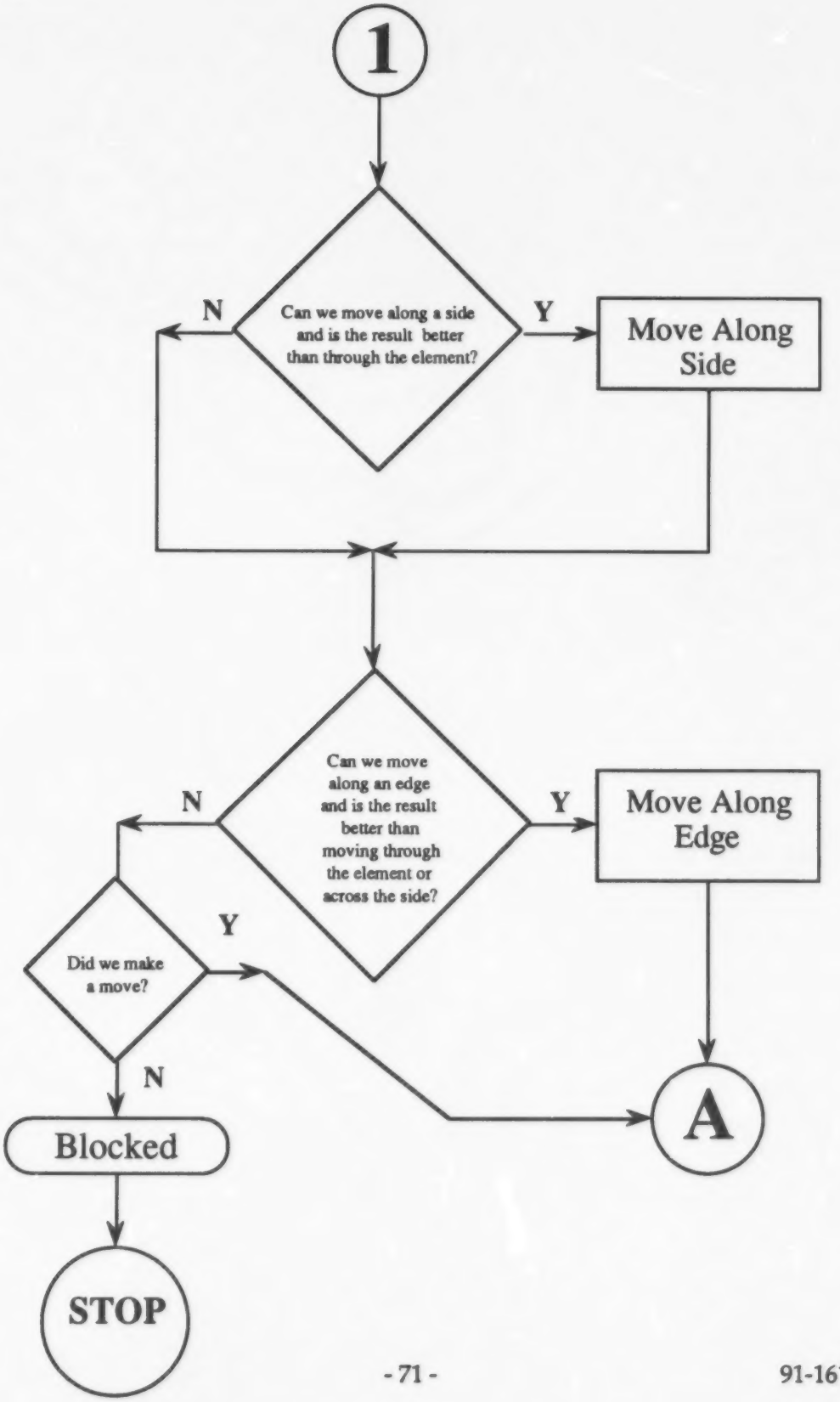
rdconfig
type: subroutine;
inputs: none;
outputs: none;
description: reads input and output filenames from configuration file;
invokes: none;
invoked by: main.

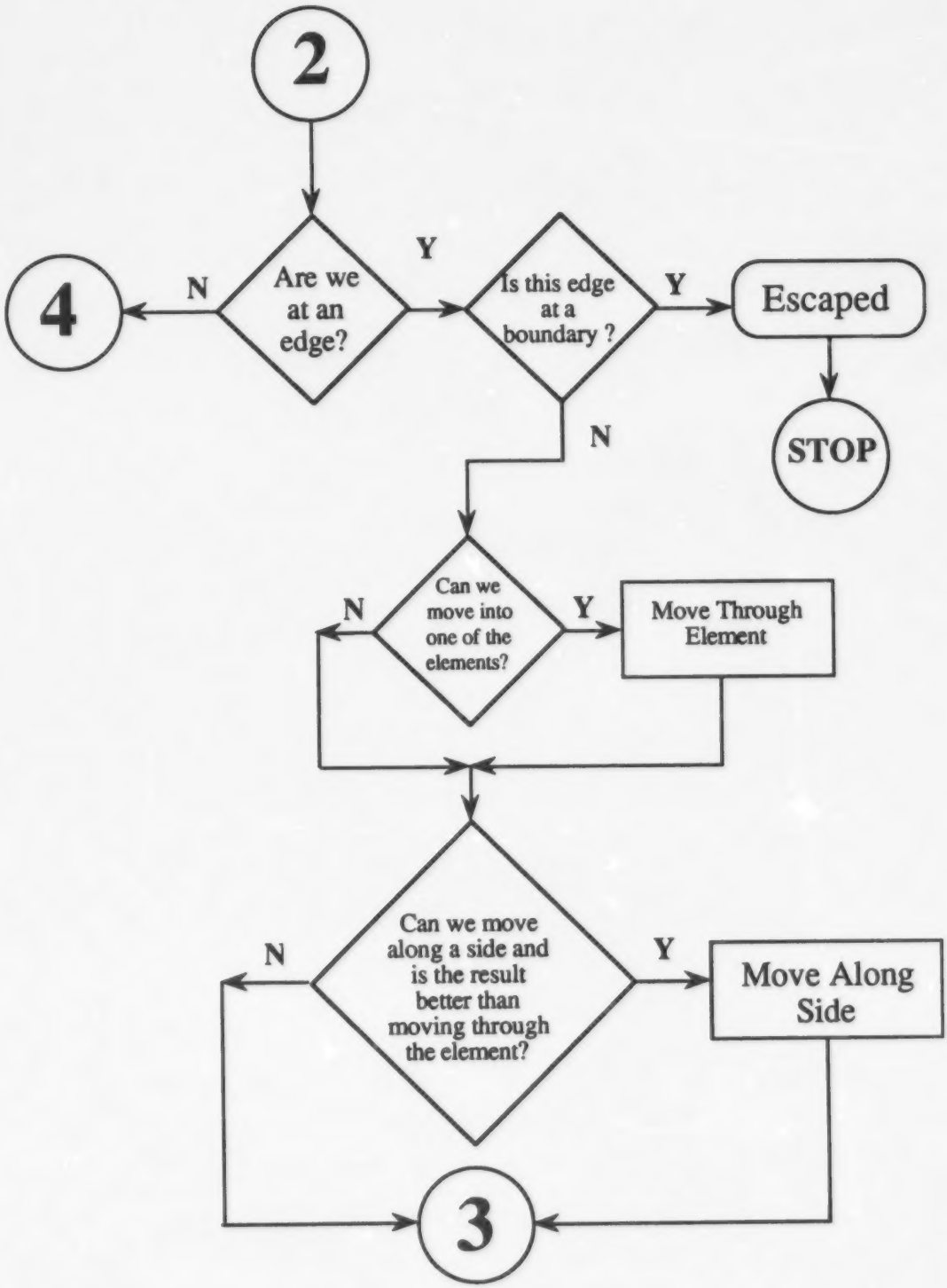
Appendix III.a.

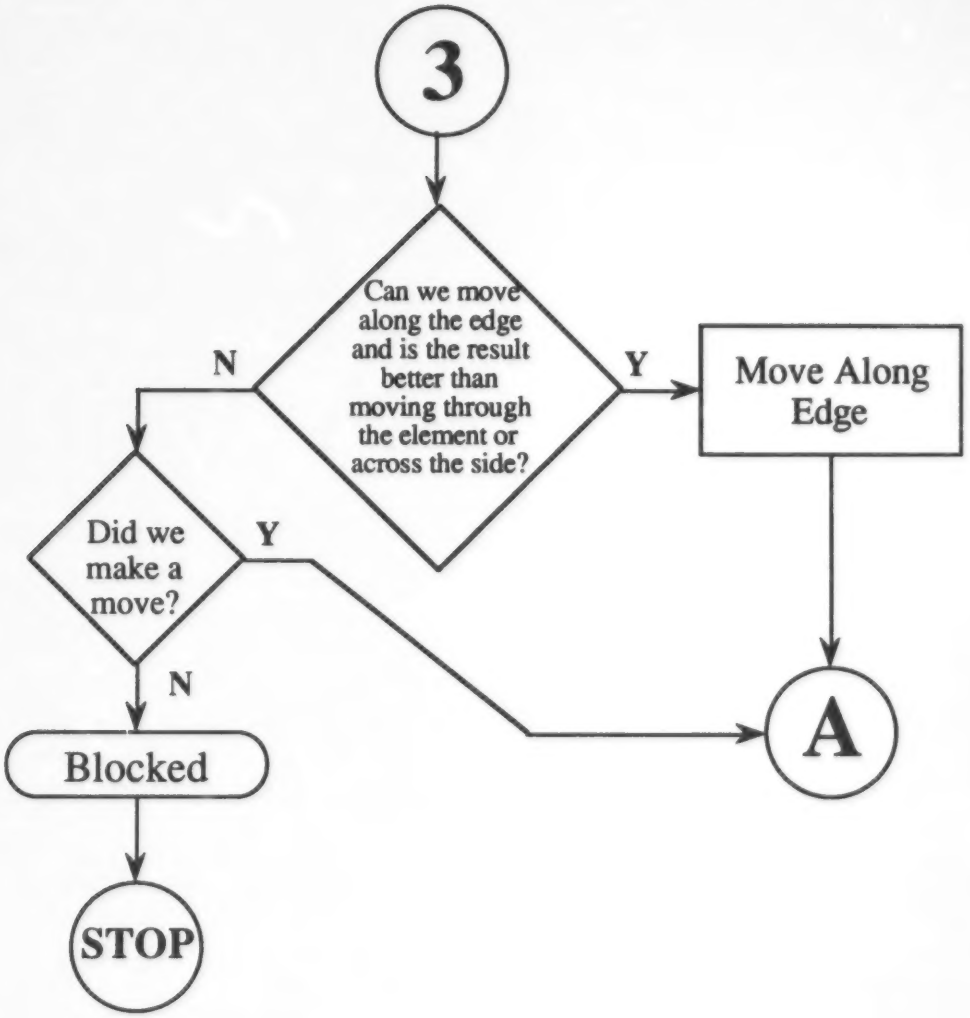
Algorithm Flow Chart.

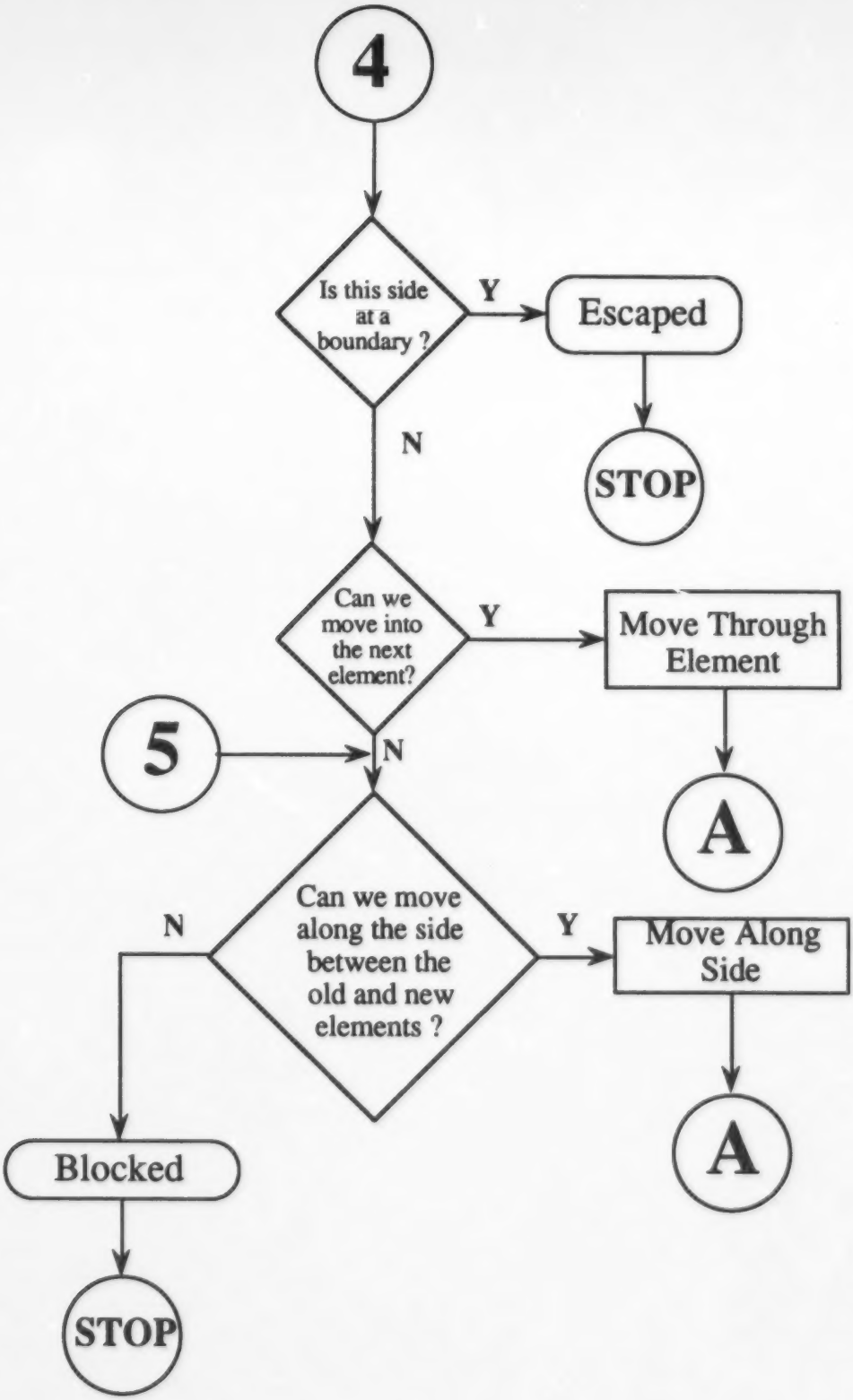

START
Can we move through the current element
N
Can we move through the adjacent element?
N
5
Y
Y
Move Through Element
A
Record Details
2
N
Are we at a Node?
Y
Is this node at an outside boundary?
Y
Escaped
STOP
N
N
Can we move into one of the elements?
Y
Move Throu Element
1

91-161


1
Can we move along a side and is the result better than through the element?
N
Y
Move Along Side
Can we move along an edge and is the result better than moving through the element or across the side?
N
Y
Move Along Edge
Did we make a move?
Y
N
Blocked
A
STOP


2
Are we at an edge?
N
4
Y
Is this edge at a boundary ?
Y
Escaped
STOP
N
Can we move into one of the elements?
N
Y
Move Through Element
Can we move along a side and is the result better than moving through the element?
N
Y
Move Along Side
3


3
Can we move along the edge and is the result better than moving through the element or across the side?
N
Y
Move Along Edge
Did we make a move?
Y
N
A
Blocked
STOP


4
Is this side at a boundary ?
Y
Escaped
STOP
N
Can we move into the next element?
Y
Move Through Element
A
5
N
Can we move along the side between the old and new elements ?
N
Y
Move Along Side
A
Blocked
STOP

Appendix III.b.

Input and Output File Formats

sample.cfg

```
elements1.dat
velocity1.dat
nodelist1.dat
nodes1.dat
summary1
detail1
100m.init
```

sample.geo

```
     Fortran format (3(D10.3))
                    (4(I5),4X,8(I5))

C*17378
    1    914.665    350.000    257.125
    2    912.320    650.000    224.250
    3    914.665    650.000    257.125
    4   1020.682    650.000    248.500
    5   1019.510    650.000    209.000
    6    912.320    350.000    224.250
    7   1020.682    350.000    248.500
    8   1019.510    350.000    209.000
    9    801.160    650.000    226.157
                    .
                    .
                    .
17369 -1117.610      3.730    214.487
17370 -1117.610     -3.730    214.487
17371 -1110.150      3.730    233.246
17372 -1110.150     -3.730    233.246
17373 -1117.610      3.730    235.037
17374 -1117.610     -3.730    235.037
17375 -1110.150      3.730    260.000
17376 -1110.150     -3.730    260.000
17377 -1117.610      3.730    260.000
17378 -1117.610     -3.730    260.000
C*15586
    1    1    1   80    8173391718617165173401734117187171661734 2
    2    1    1   80    8172151720717186173391721617208171871734 1
    3    1    1   80    8173431718817167173441733917186171651734 0
    4    1    1   80    8172171720917188173431721517207171861733 9
    5    1    1   80    8173451718917168173461734317188171671734 4
    6    1    1   80    8172181721017189173451721717209171881734 3
    7    1    1   80    8173471719017169173481734517189171681734 6
    8    1    1   80    8172191721117190173471721817210171891734 5
    9    1    1   80    8173491719117170173501734717190171691734 8
                        .
                        .
                        .
                        .
15576   22   22   80    8   88   91   95   93  221  232  234  225
15577   22   22   80    8  220  231  232  221  451  456  457  452
15578   22   22   80    8  222  233  231  220  453  458  456  451
15579   22   22   80    8  229  235  233  222  455  460  458  453
15580   22   22   80    8   79   80   27   24   99  100   92   89
15581   22   22   80    8   99  100   92   89  229  235  233  222
15582   22   22   80    8   20   25   26   21   87   90   91   88
15583   22   22   80    8   24   27   25   20   89   92   90   87
15584   22   22   80    8   87   90   91   88  220  231  232  221
15585   22   22   80    8   89   92   90   87  222  233  231  220
15586   23   23   80    8 9323 8449 8451 9324 9304 8431 8433 9305
```

sample.in

```
W3DW_100_10kc.RES
W3DW_100.GEO
   53
w3dw-350_100_10k_1.05
w3dw-350_100_10k_2.05
w3dw-350_100_10k_3.05
          .
          .
          .
w3dw-350_100_10k_53.05
   195.830 -1000.000   -25.875
   195.830  -975.000   -25.875
   195.830  -950.000   -25.875
                .
                .
                .
   195.830    75.000   -25.875
   195.830   100.000   -25.875
   195.830   125.000   -25.875
   195.830   150.000   -25.875
   195.830   175.000   -25.875
   195.830   200.000   -25.875
   195.830   225.000   -25.875
   195.830   250.000   -25.875
   195.830   275.000   -25.875
   195.830   300.000   -25.875
```

sample.ini

Fortran format (I7,F8.3,2(F10.3))

```
8813 195.830 -1000.000   -25.875
8844 195.830  -975.000   -25.875
8844 195.830  -950.000   -25.875
8844 195.830  -925.000   -25.875
8844 195.830  -900.000   -25.875
8844 195.830  -875.000   -25.875
8844 195.830  -850.000   -25.875
8844 195.830  -825.000   -25.875
8844 195.830  -800.000   -25.875
8844 195.830  -775.000   -25.875
8844 195.830  -750.000   -25.875
8844 195.830  -725.000   -25.875
8844 195.830  -700.000   -25.875
8844 195.830  -675.000   -25.875
8844 195.830  -650.000   -25.875
8910 195.830  -625.000   -25.875
8910 195.830  -600.000   -25.875
8910 195.830  -575.000   -25.875
8910 195.830  -550.000   -25.875
8910 195.830  -525.000   -25.875
8910 195.830  -500.000   -25.875
8910 195.830  -475.000   -25.875
8910 195.830  -450.000   -25.875
8910 195.830  -425.000   -25.875
8910 195.830  -400.000   -25.875
8910 195.830  -375.000   -25.875
8910 195.830  -350.000   -25.875
8921 195.830  -325.000   -25.875
8921 195.830  -300.000   -25.875
8921 195.830  -275.000   -25.875
8921 195.830  -250.000   -25.875
8921 195.830  -225.000   -25.875
8921 195.830  -200.000   -25.875
8921 195.830  -175.000   -25.875
8921 195.830  -150.000   -25.875
8921 195.830  -125.000   -25.875
8921 195.830  -100.000   -25.875
8921 195.830   -75.000   -25.875
8921 195.830   -50.000   -25.875
8936 195.830   -25.000   -25.875
8936 195.830     0.000   -25.875
8936 195.830    25.000   -25.875
8936 195.830    50.000   -25.875
8943 195.830    75.000   -25.875
8943 195.830   100.000   -25.875
8943 195.830   125.000   -25.875
8943 195.830   150.000   -25.875
8943 195.830   175.000   -25.875
8943 195.830   200.000   -25.875
```

sample.ini (cont'd)

```
8943 195.830   225.000   -25.875
8943 195.830   250.000   -25.875
8943 195.830   275.000   -25.875
8943 195.830   300.000   -25.875
```

sample.res

```
 0.0000000+000
 0.2883265+003  0.2754928+003  0.2746339+003  0.2746380+003  0.2755026+003
 0.2866106+003  0.2865436+003  0.2850214+003  0.2757257+003  0.2745818+003
 0.2864783+003  0.2883342+003  0.2740000+003  0.2740000+003  0.2900000+003
 0.2880000+003  0.2740000+003  0.2900000+003  0.2879182+003  0.2889142+003
 0.2871227+003  0.2862437+003  0.2881078+003  0.2890602+003  0.2900000+003
 0.2880000+003  0.2900000+003  0.2739454+003  0.2739389+003  0.2739429+003
 0.2739296+003  0.2758110+003  0.2758628+003  0.2738350+003  0.2738767+003
 0.2762410+003  0.2739629+003  0.2739380+003  0.2738831+003  0.2857739+003
 0.2841138+003  0.2853723+003  0.2740000+003  0.2740000+003  0.2740000+003
 0.2731407+003  0.2733150+003  0.2737256+003  0.2745253+003  0.2730000+003
 0.2730000+003  0.2846585+003  0.2834369+003  0.2860000+003  0.2734255+003
 0.2749786+003  0.2827929+003  0.2873081+003  0.2856173+003  0.2873403+003
 0.2853337+003  0.2846456+003  0.2860000+003  0.2842311+003  0.2736990+003
 0.2737715+003  0.2760836+003  0.2767860+003  0.2739518+003  0.2754160+003
      .              .              .              .              .
      .              .              .              .              .
      .              .              .              .              .
      .              .              .              .              .
      .              .              .              .              .
-0.2151720-006 -0.1551776-007 -0.4402411-007 -0.2085485-006  0.1775201-007
 0.4343965-009 -0.1779841-006  0.1607321-007 -0.2992185-008 -0.2183135-006
 0.1271995-008  0.4128549-009 -0.1989643-006 -0.4207532-009 -0.3892549-008
-0.2622241-006 -0.9540154-009  0.2769559-009 -0.1894777-006 -0.4149774-008
-0.5713549-008 -0.2815903-006  0.2253321-007  0.1393319-008 -0.1712003-006
 0.2151755-007  0.1296109-008 -0.2410256-006  0.1696799-009  0.1459228-008
-0.2755724-006 -0.1723313-008  0.2035805-008 -0.3222498-006  0.1396888-007
-0.2367751-007 -0.1402311-006  0.2280595-007  0.8389709-010 -0.1213962-006
 0.2276299-007 -0.9740462-009 -0.1545145-006  0.1260801-007 -0.2983839-007
-0.2031148-006 -0.1536112-007 -0.4046664-007 -0.2308689-006 -0.1558068-007
-0.5439569-007 -0.2024246-006 -0.7745380-009  0.2059992-009 -0.2159335-006
-0.2114048-008 -0.2118967-008 -0.3221078-006  0.2217291-007  0.2223251-009
-0.1841962-006  0.4956466-009  0.2322048-009 -0.2038199-006  0.2152357-007
-0.1009077-008 -0.2230131-006  0.4830437-010 -0.1304492-008 -0.2614680-006
-0.4766513-011  0.1047422-010  0.1915643-010
```

elements.dat

Fortran format (9(I6))

1	17339	17186	17165	17340	17341	17187	17166	17342
2	17215	17207	17186	17339	17216	17208	17187	17341
3	17343	17188	17167	17344	17339	17186	17165	17340
4	17217	17209	17188	17343	17215	17207	17186	17339
5	17345	17189	17168	17346	17343	17188	17167	17344
6	17218	17210	17189	17345	17217	17209	17188	17343
7	17347	17190	17169	17348	17345	17189	17168	17346
8	17219	17211	17190	17347	17218	17210	17189	17345
9	17349	17191	17170	17350	17347	17190	17169	17348
10	17220	17212	17191	17349	17219	17211	17190	17347
				.				
				.				
				.				
				.				
15577	220	231	232	221	451	456	457	452
15578	222	233	231	220	453	458	456	451
15579	229	235	233	222	455	460	458	453
15580	79	80	27	24	99	100	92	89
15581	99	100	92	89	229	235	233	222
15582	20	25	26	21	87	90	91	88
15583	24	27	25	20	89	92	90	87
15584	87	90	91	88	220	231	232	221
15585	89	92	90	87	222	233	231	220
15586	9323	8449	8451	9324	9304	8431	8433	9305

nodelist.dat

Fortran format (2(I5))
(I7)

```
  1    9
15518
15519
15522
15523
15555
15556
15571
15572
  2    9
15393
15394
15400
15402
15480
15481
15518
15519
  3    9
15400
15401
15402
15408
15518
15519
15522
15523
  .
  .
  .
  .
```

nodelist.dat

Fortran format (2(I5))
(I7)

```
  1    9
15518
15519
15522
15523
15555
15556
15571
15572
  2    9
15393
15394
15400
15402
15480
15481
15518
15519
  3    9
15400
15401
15402
15408
15518
15519
15522
15523
  .
  .
  .
  .
```

nodes.dat

Fortran format (I5,3(D10.3))

```
    1   914.665    350.000    257.125
    2   912.320    650.000    224.250
    3   914.665    650.000    257.125
    4  1020.682    650.000    248.500
    5  1019.510    650.000    209.000
    6   912.320    350.000    224.250
    7  1020.682    350.000    248.500
    8  1019.510    350.000    209.000
    9   801.160    650.000    226.157
   10   776.333    650.000    258.078
                      .
                      .
                      .
                      .
17369 -1117.610      3.730    214.487
17370 -1117.610     -3.730    214.487
17371 -1110.150      3.730    233.246
17372 -1110.150     -3.730    233.246
17373 -1117.610      3.730    235.037
17374 -1117.610     -3.730    235.037
17375 -1110.150      3.730    260.000
17376 -1110.150     -3.730    260.000
17377 -1117.610      3.730    260.000
17378 -1117.610     -3.730    260.000
```

Detailed output

PATHLINE COORDINATES

x (m)	y (m)	z (m)	Element #	atedge	atnode	Accum. Travel Time (s)	Accum. Travel Distance
195.83	-1000.00	-25.88	8813	T	F	0.0000D+00	0.0000D+00
120.29	-980.69	0.30	8839	F	F	0.1988D+10	0.8224D+02
45.00	-953.07	27.11	8836	F	F	0.4117D+10	0.1668D+03
-79.00	-893.67	68.79	8830	F	F	0.7430D+10	0.3105D+03
-202.41	-820.05	111.49	8826	F	F	0.1045D+11	0.4604D+03
-240.75	-746.42	130.55	9564	F	F	0.1269D+11	0.5456D+03
-240.26	-744.61	155.72	10295	F	F	0.1304D+11	0.5708D+03
-239.65	-744.21	166.29	10286	F	F	0.1319D+11	0.5814D+03
-239.32	-743.31	175.49	10995	F	F	0.1334D+11	0.5907D+03
-238.45	-741.83	197.40	11109	F	F	0.1367D+11	0.6126D+03
-237.88	-741.19	212.00	11110	F	F	0.1391D+11	0.6273D+03
-237.06	-740.02	229.68	11869	F	F	0.1417D+11	0.6450D+03
-236.91	-739.73	238.19	11870	F	F	0.1431D+11	0.6535D+03
-236.84	-739.16	255.14	0	F	F	0.1460D+11	0.6705D+03

Particle escaped...simulation ended

Summary output

PARTICLE TRACKING RESULTS

Element number	Initial location x (m)	y (m)	z (m)	Exit location x (m)	y (m)	z (m)	Travel Time (years)	Status
4386	837.68	-1000.00	-239.40	-213.52	-922.40	255.54	1252.5	escaped
6419	837.68	-950.00	-239.40	-216.43	-915.61	255.54	1192.1	escaped
6419	837.68	-900.00	-239.40	-217.70	-896.98	255.49	1133.7	escaped
6419	837.68	-850.00	-239.40	-217.71	-846.98	255.35	1133.7	escaped
6419	837.68	-800.00	-239.40	-217.71	-796.99	255.21	1133.6	escaped
6419	837.68	-750.00	-239.40	-217.72	-746.99	255.06	1133.6	escaped
6419	837.68	-700.00	-239.40	-217.72	-697.00	254.92	1133.5	escaped
6419	837.68	-650.00	-239.40	-222.45	-650.05	254.81	1132.3	escaped
6401	837.68	-600.00	-239.40	-247.30	-650.02	254.94	1117.5	escaped
6401	837.68	-550.00	-239.40	-252.16	-650.01	254.97	1084.8	escaped
6401	837.68	-500.00	-239.40	-252.51	-650.01	254.97	1045.3	escaped
6401	837.68	-450.00	-239.40	-253.17	-650.01	254.97	1037.7	escaped
6401	837.68	-400.00	-239.40	-302.08	-626.79	255.04	1085.7	escaped
6401	837.68	-350.00	-239.40	-302.50	-577.54	255.12	1086.1	escaped
6383	837.68	-300.00	-239.40	-302.67	-528.87	255.20	1086.3	escaped
6383	837.68	-250.00	-239.40	-308.54	-504.81	255.24	1093.2	escaped
6383	837.68	-200.00	-239.40	-307.19	-472.89	255.30	1091.1	escaped
6383	837.68	-150.00	-239.40	-306.53	-444.31	255.34	1088.6	escaped
6383	837.68	-100.00	-239.40	-306.58	-418.70	255.39	1085.3	escaped
6383	837.68	-50.00	-239.40	-306.40	-396.95	255.42	1079.7	escaped
6365	837.68	0.00	-239.40	-292.86	-372.18	255.46	42873.7	escaped
6374	837.68	50.00	-239.40	-362.02	-354.39	255.49	1123.8	escaped
6374	837.68	100.00	-239.40	-430.85	-332.34	255.25	1175.9	escaped
6374	837.68	150.00	-239.40	-395.51	-293.25	255.40	1155.2	escaped
6374	837.68	200.00	-239.40	-395.04	-262.96	255.40	1161.6	escaped
6374	837.68	250.00	-239.40	-398.23	-234.86	255.39	1170.6	escaped
6374	837.68	300.00	-239.40	-400.56	-207.82	255.38	1178.3	escaped
6392	837.68	350.00	-239.40	-405.20	-188.42	255.36	1199.3	escaped
6392	837.68	400.00	-239.40	-401.67	-167.49	255.38	1210.8	escaped
6392	837.68	450.00	-239.40	-392.62	-146.83	255.41	1217.3	escaped
6392	837.68	500.00	-239.40	-381.89	-127.10	255.46	1221.4	escaped
6392	837.68	550.00	-239.40	-380.29	-112.80	255.47	1236.2	escaped
6392	837.68	600.00	-239.40	-380.86	-99.37	255.47	1252.0	escaped
6410	837.68	650.00	-239.40	-377.34	-87.12	255.48	1277.5	escaped
6410	837.68	700.00	-239.40	-371.07	-74.73	255.51	1298.3	escaped
6410	837.68	750.00	-239.40	-362.00	-61.96	255.55	1313.0	escaped
6410	837.68	800.00	-239.40	-360.60	-50.00	255.55	1323.0	blocked
6410	837.68	850.00	-239.40	-367.29	-36.27	255.52	1353.0	escaped
6410	837.68	900.00	-239.40	-370.10	-23.73	255.51	1384.8	escaped
6410	837.68	950.00	-239.40	-372.62	-12.25	255.50	1415.5	escaped
1007	837.68	1000.00	-239.40	-375.54	-4.69	255.49	1457.1	escaped

APPENDIX IV

User's Manual

3D Particle Tracking System

1. Introduction

The 3D Particle Tracking system is intended for use in tracking radio-nuclides that escape from an underground nuclear waste repository. It is important, when planning such a repository, to know how long it will take a particle, that escapes containment, to reach the surface and enter the biosphere. By simulating the motion of an escaping particle through a rock mass, an estimate of the time can obtained.

The program traces the motion of a particle through a finite element mesh that represents a mass of ground rock. As the particle moves through the mesh, its progress is reported. The progress report at the end of each move includes the start and finish coordinates and the cumulative travel time.

For the operation of the program, it is assumed that there exists a compatible finite-element mesh that correctly models the rock for the area in question. There should be, for each element, the node coordinates and average vector velocity component (of the hydraulic gradient existing within the element). Any Cartesian coordinate system may be used, with any origin, as long as the distance scale and velocity units are consistent with the time units expected. Refer to the System Manual for required input/output files and their dependencies.

The purpose of the User Manual for the 3D Particle Tracking System is to provide a comprehensive, step-by-step, guide to correctly run the system. The scope of this document includes: system requirements and user procedure.

2. System Requirements

The particle tracking system is written entirely in standard FORTRAN 77. Therefore, a compatible compiler is required to generate a working system in a new environment. The memory and disk storage requirements vary with the size of the mesh, but one of 15000 elements can be operated with nine (9) megabytes of RAM and four (4) megabytes of disk space.

Further information pertaining to the system requirements is discussed in the *System Manual*.

3. Assumptions

The User Manual assumes the following:

- The user understands and is familiar with the project;

_System requirements, as outlined, are met;
_The user has read section *4.2 File Dependencies and Inputs/Outputs* in the *System Manual*;

4. Operating Procedure

The operating procedure assumes the system is to be used for a new test problem supplied by AECL. Several of the steps may be skipped when running the system on existing test problems. Care must be taken, when running existing test problems, to ensure all necessary files exist.

Sample input/output data files are provided in Appendix III.b of the *System Manual.*

The operating procedure is as follows:

1. Velocity Distribution File

The AECL file <filename>.res must be modified to generate the file velocity<x>.dat. At the present time Ontario Hydro does not have enough information concerning the format of the file to automate this conversion. Therefore, the user may have to write a small FORTRAN program to accomplish this task. First, remove the first two (2) blocks of data corresponding to the nodal and head values. Second, the format of the numbers must be changed to a legitimate FORTRAN format. Each value must be changed by inserting a 'D' character between the base and the exponent. For example, -1.2345678+001 must be changed to -1.2345678D+001. Third, the velocity<x>.dat file must be formatted with three (3) values per line rather than five (5) values per line.

2. Geometry File and the Utility Program

The AECL file <filename>.geo must be processed by the **ptutil** program to produce the files nodes<x>.dat, elements<x>.dat, and nodelist<x>.dat. To generate these files, run the executable ptutil.exe. The user will be prompted for the pathname of the .geo file and the pathname of the appropriate configuration file (see step 4). The program will then process these files and terminate in a few moments. Check to see that the three .dat files now exist.

3. Start Coordinate Input File

The starting coordinates and element ID's for pathway analysis must be set up in a file created by the user. The starting coordinates information can be found in AECL's file <filename>.in. In addition, for each point, the id number of the element within which the starting point falls must precede the point's coordinates (Ontario Hydro does not presently have the information concerning the corresponding element numbers that would be required to automate the process of adding this data to the .init file). The final file should have four (4) columns: element ID, x-coordinate, y-coordinate, z-coordinate. This file should be saved with a .init extension. An example .init file is given in Appendix III.b of the *System Manual.*

4. Setup File Creation

A configuration file, containing the input/output files to be used by the system, must be created by the user. The format of the file is as follows:

elements<x>.dat {element data file created by program ptutil.exe}

velocity<x>.dat	{velocity file created in step 1}
nodelist<x>.dat	{node list created by program ptutil.exe}
nodes<x>.dat	{node file created by program ptutil.exe}
summary<x>	{summary output file generated by program}
detail<x>	{detailed pathway file generated by program}
w3dw100_0kc.in	{start coordinate input file created in step 3)

5. Program Compilation

Look at the file, max.f, and check to see if the two (2) constants: MAXNODES and MAXELEMS are set to the number of nodes in the problem and the number of elements in the problem respectively. These numbers should be the same as in the AECL .geo file. If the constants are not correct, edit this file accordingly.

A UNIX Makefile has been included to compile the particle tracking program. Look at this file and use the corresponding commands to compile the program in a non-UNIX environment. Remember that there is a total of four files which must be compiled and loaded together.

6. Program Execution

Now (finally!), to run the program, type: ptsim.exe. The program will prompt you for the name of a configuration file. Enter the name of the file (you should have created this in step 4).
The program should now be running. Upon completion, two output files will be generated as named in the configuration file, probably detail<x> and summary<x>. Appendix IV.a contains sample detail and summary files.

APPENDIX IV.a

Sample Output Files

PARTICLE TRACKING RESULTS

Element number	Initial location x (m)	y (m)	z (m)	Exit location x (m)	y (m)	z (m)	Travel Time (years)	Status
4386	837.68	-1000.00	-239.40	-213.52	-922.40	255.54	1252.5	escaped
6419	837.68	-950.00	-239.40	-216.43	-915.61	255.54	1192.1	escaped
6419	837.68	-900.00	-239.40	-217.70	-896.98	255.49	1133.7	escaped
6419	837.68	-850.00	-239.40	-217.71	-846.98	255.35	1133.7	escaped
6419	837.68	-800.00	-239.40	-217.71	-796.99	255.21	1133.6	escaped
6419	837.68	-750.00	-239.40	-217.72	-746.99	255.06	1133.6	escaped
6419	837.68	-700.00	-239.40	-217.72	-697.00	254.92	1133.5	escaped
6419	837.68	-650.00	-239.40	-223.11	-650.05	254.81	1121.5	escaped
6401	837.68	-600.00	-239.40	-247.30	-650.02	254.94	1113.8	escaped
6401	837.68	-550.00	-239.40	-252.16	-650.01	254.97	1082.5	escaped
6401	837.68	-500.00	-239.40	-252.51	-650.01	254.97	1043.1	escaped
6401	837.68	-450.00	-239.40	-253.17	-650.01	254.97	1035.7	escaped
6401	837.68	-400.00	-239.40	-302.08	-626.79	255.04	1085.7	escaped
6401	837.68	-350.00	-239.40	-302.50	-577.54	255.12	1086.1	escaped
6383	837.68	-300.00	-239.40	-302.67	-528.87	255.20	1086.3	escaped
6383	837.68	-250.00	-239.40	-308.54	-504.81	255.24	1093.2	escaped
6383	837.68	-200.00	-239.40	-307.19	-472.89	255.30	1091.1	escaped
6383	837.68	-150.00	-239.40	-306.53	-444.31	255.34	1088.6	escaped
6383	837.68	-100.00	-239.40	-306.58	-418.70	255.39	1085.3	escaped
6383	837.68	-50.00	-239.40	-306.40	-396.95	255.42	1079.7	escaped
6365	837.68	0.00	-239.40	-292.86	-372.18	255.46	42873.7	escaped
6374	837.68	50.00	-239.40	-362.02	-354.39	255.49	1042.9	escaped
6374	837.68	100.00	-239.40	-430.45	-332.34	255.25	1174.3	escaped
6374	837.68	150.00	-239.40	-395.51	-293.25	255.40	1155.2	escaped
6374	837.68	200.00	-239.40	-395.04	-262.96	255.40	1161.6	escaped
6374	837.68	250.00	-239.40	-398.23	-234.86	255.39	1170.6	escaped
6374	837.68	300.00	-239.40	-400.56	-207.82	255.38	1178.3	escaped
6392	837.68	350.00	-239.40	-405.20	-188.42	255.36	1197.7	escaped
6392	837.68	400.00	-239.40	-401.67	-167.49	255.38	1210.8	escaped
6392	837.68	450.00	-239.40	-392.62	-146.83	255.41	1217.3	escaped
6392	837.68	500.00	-239.40	-381.89	-127.10	255.46	1221.4	escaped
6392	837.68	550.00	-239.40	-380.29	-112.80	255.47	1236.2	escaped
6392	837.68	600.00	-239.40	-380.86	-99.37	255.47	1182.8	escaped
6410	837.68	650.00	-239.40	-377.34	-87.12	255.48	1277.5	escaped
6410	837.68	700.00	-239.40	-371.07	-74.73	255.51	1298.3	escaped
6410	837.68	750.00	-239.40	-362.00	-61.96	255.55	1313.0	escaped
6410	837.68	800.00	-239.40	-360.60	-50.00	255.55	1323.0	escaped
6410	837.68	850.00	-239.40	-367.29	-36.27	255.52	1237.8	escaped
6410	837.68	900.00	-239.40	-370.10	-23.73	255.51	1384.8	escaped
6410	837.68	950.00	-239.40	-372.62	-12.25	255.50	1415.4	escaped
1007	837.68	1000.00	-239.40	-375.54	-4.69	255.49	1457.1	escaped

PATHLINE COORDINATES

x (m)	y (m)	z (m)	Element #	atedge	atnode	Accumulated Travel Time (s)
837.68	-1000.00	-239.40	4386	false	false	0.00e+000
668.38	-1033.83	-183.69	8119	false	false	7.46e+009
590.50	-1041.18	-157.24	8118	false	false	1.08e+010
513.14	-1044.52	-129.80	8134	false	false	1.39e+010
468.82	-1044.56	-115.21	8157	false	false	1.56e+010
424.56	-1043.13	-100.39	8178	false	false	1.74e+010
376.97	-1039.92	-85.04	8208	false	false	1.93e+010
329.46	-1035.13	-69.47	8235	false	false	2.12e+010
263.43	-1025.91	-47.25	8260	false	false	2.37e+010
197.95	-1013.88	-24.56	8283	false	false	2.60e+010
143.91	-1000.00	-5.70	8858	false	false	2.80e+010
121.53	-998.83	2.07	8879	false	false	2.86e+010
45.32	-990.77	28.88	8880	false	false	3.09e+010
-78.32	-970.40	70.91	8881	false	false	3.42e+010
-201.45	-944.53	113.33	8882	false	false	3.69e+010
-217.74	-924.86	123.31	8862	false	false	3.80e+010
-216.49	-924.37	147.11	8841	false	false	3.82e+010
-215.70	-924.29	157.90	8842	false	false	3.83e+010
-215.12	-923.86	170.41	8818	false	false	3.85e+010
-214.53	-923.73	189.94	8819	false	false	3.87e+010
-213.78	-922.99	208.34	8792	false	false	3.89e+010
-213.94	-923.09	222.82	8793	false	false	3.91e+010
-213.18	-922.33	243.09	8763	false	false	3.93e+010
-213.52	-922.40	255.54	0	false	false	3.95e+010

Particle escaped ... simulation ended.